Vidyalankar

T.Y. B.Sc. (IT) : Sem. VI

Software Quality Assurance

Unit II : Fundamentals of Testing

SYLLABUS **Weightage : 15 Marks**

Fundamentals of Testing : Introduction, Necessity of testing, What is testing? Fundamental test process, The psychology of testing, Historical Perspective of Testing, Definitions of Testing, Approaches to Testing, Testing During Development Life Cycle, Requirement Traceability Matrix, Essentials of Software Testing, Workbench, Important Features of Testing Process, Misconceptions About Testing, Principles of Software Testing, Salient Features of Good Testing, Test Policy, Test Strategy or Test Approach, Test Planning, Testing Process and Number of Defects Found in Testing, Test Team Efficiency, Mutation Testing, Challenges in Testing, Test Team Approach, Process Problems Faced by Testing, Cost Aspect of Testing, Establishing Testing Policy, Methods, Structured Approach to Testing, Categories of Defect, Defect, Error, or Mistake in Software, Developing Test Strategy, Developing Testing Methodologies (Test Plan), Testing Process, Attitude Towards Testing (Common People Issues), Test Methodologies/Approaches, People Challenges in Software Testing, Raising Management Awareness for Testing, Skills Required by Tester, Testing throughout the software life cycle, Software development models, Test levels, Test types, the targets of testing, Maintenance testing.

SQA - U2

Q.1 What is the role of testing in software development, maintenance and operations?

Ans.:

- The human errors can cause a defect or fault to be introduced at any stage within the software development life cycle, and the consequences can be damaging.
- Rigorous testing is necessary during development and maintenance to identify defects, in order to reduce failures in the operational environment.
- This includes looking for places in the user interface where a user might make a mistake in input of data or in the interpretation of the output, and looking for potential weak points for intentional and malicious attack.
- Executing tests helps us move towards improved quality of product and service, but that is just one of the verification and validation methods applied to products.
- Processes are also checked, for example by audit. A variety of methods may be used to check work, some of which are done by the author of the work and some by the others to get an independent view.

Q.2 What is software testing? Explain the objectives of testing.

Ans.:

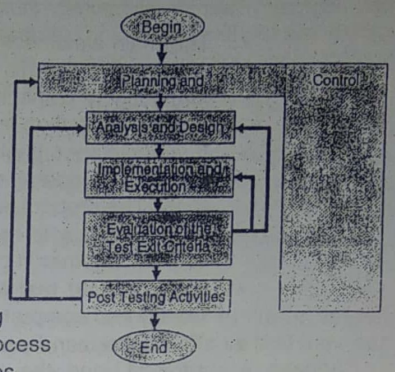
- Software testing is an umbrella activity in which the developed software is tested with the intention of finding an error, bug, faults or failures present in the software.
- In software testing the developed software is executed to determine whether all requirements of customer must be satisfied or not.
- In software testing the developed software is executed to verify the actual and expected result.
- Software Testing has different goals and objectives. The major objectives of Software testing are as follows :
  - Finding defects which may get created by the programmer while developing the software.
  - Gaining confidence in and providing information about the level of quality.
  - To prevent defects.
  - To make sure that the end result meets the business and user requirements.
  - To ensure that it satisfies the BRS that is Business Requirement Specification and SRS that is System Requirement Specifications.
  - To gain the confidence of the customers by providing them a quality product.
- Software testing helps in finalizing the software application or product against business and user requirements. It is very important to have good test coverage in order to test the software application completely and make it sure that it's performing well and as per the specifications.

Q.3 Explain the phases involved in Fundamental test process.

Ans.: The Fundamental of Test Process are :

1. Test Planning and Control
2. Test Analysis and Design
3. Test implementation and execution

4. Evaluating exit criteria and Reporting
5. Test closure activities



1. Test Planning and Control :

Test Planning is a Fundamental Test Process which defining the objective and goal of the testing process. Using requirement specification test plan will make more detailed. It is the continuous process and performed in all life cycles.

- Deciding the scope and risk of testing
- Deciding the overall approach of testing
- Scheduling test analysis and design process
- Assigning resources to different activities

2. Test Analysis and Design :

Test Analysis and Design is a Fundamental Test Process which creating test conditions & test cases. In this process performed major tasks like reviewing the test basis, identifying the test conditions based on analysis of test items, writing test cases, identifying necessary test data to support the test conditions and test cases, Designing the test environment.

3. Test Implementation and Execution :

It is a Fundamental Test Process in which actual work is done. In this process executing test cases with test data.

- Creating test suites from test executions
- Checking test environments, updating traceability between test basis and test cases
- Executing test procedures using test execution tools
- Checking actual result with expected results
- Reporting errors & creating Incident reports

4. Evaluating exit criteria and Reporting :

Evaluating exit criteria is a process defining when to stop testing. It depends on the coverage of code, functionality or risk. Basically, it also depends on business risk, cost and time.

5. Test Closure Activities :

Test closure activities are the last process in the fundamental test process. In this process collect data from completed test process and test wares.

- Ensure that deliverable has been delivered or not
- Closing incident report
- Documenting all the systems
- Archiving all the testware, test environment and infrastructure

Q.4 Why is independent testing necessary?
OR
What is the psychology of independent testing?

Ans.:
- The mindset to use while testing and reviewing is different from the one we use while analyzing or developing.

- The main weakness of developer tests is that developers who have to test their own programs will tend to be too optimistic. There is the danger of forgetting reasonable test cases or, because they are more interested in programming than in testing, only testing superficially.
- If a developer implemented a fundamental design error—for example, if he/she misunderstood the task—then he/she will not find this using her own tests. The proper test case will not even come to mind.
- One possibility to decrease this problem of "blindness to one's own errors" is to work together in pairs and let a colleague test the programs.
- ***Independent test team*** : An independent testing team is beneficial for test quality and comprehensiveness.
- The tester can look at the test object without bias. It is not the tester's own product, and the tester does not necessarily share possible developer assumptions and misunderstandings.
- The tester must, however, acquire the necessary knowledge about the test object in order to create test cases, which takes time.
- But the tester typically has more testing knowledge. A developer does not have this knowledge and must acquire it (or rather should have acquired it before, because the necessary time is often not unavailable during the project).
- Though programmers do test their own code, business analysts review their own requirements, but it is often difficult to find their own mistakes. So they often rely on others to help test their work who are testing specialists.
- Testing involves a succession of people each carrying out a different level of testing allowing independent test of the system.

Q.5 Explain the evolution of software testing from debugging to prevention based testing.

Ans.:

1. **Debugging-oriented testing :**
   - During initial phase, software testing was considered as a part of software development. Developers were expected to perform debugging on the application, which they were building.
   - Tests were not documented, and were mainly done in heuristic way.
2. **Demonstration-oriented testing :**
   - In this phase, there was an introduction of software testers independent of development activity. Their main aim was to show to customers that the software really works. The approach was oriented towards demonstration that the software could do something which was expected by the customers.
3. **Destruction-oriented testing :**
   - This approach was the basis of Glenford's Myer's theory of software testing. As per this approach, it was not sufficient to only test the software positively but users must also be protected from any failure of application. The tester's responsibility was proving that software does not fail at some abnormal instances.
4. **Evaluation-oriented testing :**
   - Evaluation-oriented testing is executed nowadays at many places of software development where the product as well as process of software development is evaluated. It corresponds to the testing process definition where software is

evaluated against some fixed parameters derived from quality factors (test factors).It is believed that there is no possibility of software without any defect, but some defect may be acceptable to the customer. This approach refers to level of confidence given to customer and application is evaluated against it.

5. Prevention-oriented testing :

- Prevention-based testing is done in some highly matured organizations. Testing is considered as a prevention activity where process problems are used to improve it so that defect free products can be produced. This reduces dependency on testing as a way to improve the quality of software.

Q.6 Define Testing. Why is Testing necessary?

Ans.: Testing is defined as 'execution of a work product with intent to find a defect'. The primary role of software testing is to expose hidden defects so that they can be fixed. Software testing is very important because of the following reasons :

1. Understanding of customer requirements may differ from person to person. One must challenge the understanding at each stage of development, and there must be some analysis of customer expectations.
2. Development people assume that whatever they have developed is as per customer requirements and will always work. But it is difficult to create real life scenario and undertake actual execution of a product at each level of software building.
3. Different entities are involved in different phases of software development. Their work may not be matching exactly with each other or with the requirement statements. Gaps between requirements, design may not be traceable unless testing is performed.
4. Integration issues may arise when different units do not work together, even though they work independently. Hence, there must be a testing after integration of different units.
5. There is a possibility of being blindfold where developers fail to recognize their own faults and think that code is perfect and there is no chance of improvement. Testers have to challenge each assumption and decision taken during development.

Q.7 Identify and explain any two approaches of Testing.
OR
Explain big bang approach of software testing with its characteristics.
OR
Explain total quality management approach.

Ans.: Following are the different approaches of Testing :

1. Big Bang Approach of Testing :

- Big bang approach involve testing software system after development work is completed and before releasing the software to customer.
- Testing done at the end of development cycle may show the defects pertaining to any phase of development such as requirements, design and coding.
- Phase-wise percentage of defects can be found out using this approach.

Table : Phase-wise defect distribution.

Development Phases	Percentage of Defects
Requirements	58
Design	35
Coding	5
Other	2

2. Total Quality Management Approach :

- If there is a process definition for testing software and these processes are optimized and capable , no or less defects are produced.
- This approach focuses on optimizing such processes and reducing the defect removal cost during production.

3. TQM as Big Bang Approach :

Following are the stages of very famous cost triangle i.e TQM cost triangle:

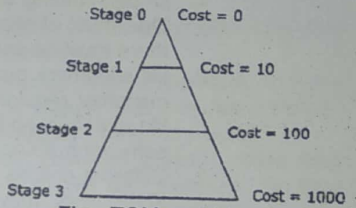


Fig.: TQM cost triangle.

- **Stage 0 :** A stage of maturity in an organization where no verification/ validation is required to certify the product quality. The cost involved is 'zero' or the benefits given by investment in process optimization and deployment make quality free.
- **Stage 1 :** An organization has very good verification process in case some defects are produced during any development stage which may detect the defects at the earliest possible stage.
  There will be a small cost of verification and fixing involved also called at appraisal cost represented by 10.
- **Stage 2 :** If some defects escape the verification process , still there are capable validation processes for filtering the defects .Cost of validation and subsequent defect fixing is much higher than verification. This cost is represented by 100.
- **Stage 3 :** At the bottom of pyramid, there is the highest cost associated with the defects found by customer during production or acceptance testing. This is represented by 1000 showing that cost paid for fixing such defect is huge.

Q.8 Explain testing during development life cycle.

Ans.: Following are the software development life cycle phases and testing associated with it.

- **Requirement testing :** Requirement Testing is used to evaluate whether all requirements are covered in requirement statement or no. This type of testing involves building use cases from the requirement statement, without any assumptions, by referring the defined requirements.
- **Design testing :** Design Testing involves testing of high-level design (system architecture) as well as low-level design (detail design) .High level design testing covers mock running of future application. For low level design system requirements and technical requirements are mapped with the entities created in design.

- **Code Testing :** Code files, Tables, stored procedures etc., are written by developers as per guidelines, standards and detail design specifications. In reality, coders do not implement requirements directly but they implement detail design as defined by the designer. Code testing / unit testing is done using stubs/drivers.
- **Test Scenario and test case testing :** Test scenarios are written by testers to address testing needs of a software application. Test cases are derived from test scenarios which are related to requirements and designs.
  Test scenarios can be functional as well as structural, depending upon the type of requirement and design they are addressing.

Q.9 What is Requirement Traceability matrix? State its advantages and disadvantages.

Ans.:
- Requirement traceability matrix is one way of doing the complete mapping for the software.
- One can expect a blueprint of an entire application using requirement traceability matrix.
- It is table which consists of Requirements, high-level design, low-level design, code files stored procedures, test scenarios, test cases and results.

Table : Requirement traceability matrix.

Requirements	High-level Design	Low-level design	Code files/ Stored Procedures/TBLs	Test scenario	Test cases	Test results

Advantages :
- Entire software development can be tracked completely through requirement traceability matrix.
- Any test case failure can be tracked through requirements, design , coding etc.
- Any changes in requirements can be affected through entire work product upto test cases and any test case failure can be traced back to requirements.
- The application becomes maintainable as one has complete relationship from requirement till test results available.

Disadvantages :
- Number of requirements is huge. It is very difficult to create requirement traceability matrix manually.
- There may be one to many, many to one and many to many relationships between various elements of traceability matrix, when we are trying to connect columns and rows of traceability and maintaining these relationships need huge efforts.
- Requirements change frequently, and one needs to update the requirement traceability matrix whenever there is a change.
- A customer may not find value in it and may not pay for it.

Q.10 Explain concept of TQM cost perceptive.

Ans.: TQM aims at reducing the cost of development and cost of quality through continual improvement. It defines the cost incurred in development and quality into three parts as follows :

1. Green Money/Cost of prevention :

- An organization may have defined processes, guidelines, standards of development, testing etc. and also define a training program to all people involved in development and testing.
- This may represent a cost of prevention.
- Creation and use of formats, templates, etc. acquiring various process models and standards, etc. also represent cost of prevention.
- This in an investment by the organization and is supposed to give returns.
- This is also termed as 'green money'. It is believed that 1 part of cost of prevention can reduce 10 parts of cost of appraisal and 100 parts of cost of failures.

2. Blue Money/Cost of appraisal :

- An organization may perform various levels of reviews and testing to appraise the quality of the product and the process followed.
- The cost incurred in first time reviews and testing is called as the cost of appraisal.
- There is no return on investment but this helps in identifying the process capabilities and process related problems, if any.
- This is termed as 'blue money' as it can be recovered from the customer.
- It is believed that1 Part of cost of appraisal can reduce 10 parts of cost of failure.

3. Red Money/Cost of Failure :

- Cost of failure starts when there is any defect or violation detected at any stage of development including post-delivery efforts spent on defect fixing.
- An extent of rework, retesting, scrapping etc. represents cost of failure.
- There may be indirect costs such as customer dissatisfaction, no repeat orders, etc. This is termed as 'red money'.
- This cost affects the profitability of the organization badly.

Q.11 Explain essentials of software testing.

Ans.: Software testing is also viewed as an exercise of doing a SWOT analysis of software product where we can build the software on the basis of strengths of the process of development and testing, and overcome weakness in the processes to the maximum extent possible :

- **Strengths :** Some areas of software are very strong, and no(very less) defects are found during testing of such areas. The areas may be in terms of some modules, screens, algorithms or processes etc. This represents strong areas.
- **Weakness :** The area of software where requirement compliance is on the verge of failure represent weak areas which needs root cause analysis. The organization may initiate training, communication etc.

- **Opportunity :** Some areas of the software which satisfy implied requirements by customer but also leaving enough space for improving it further. These are the areas of opportunities to delight the customer by improving further.
- **Threats :** Threats are the problems or defects with the software which result into failure. It can be associated with unclear requirements, knowledge base etc. An organization must invest in making these processes stronger.

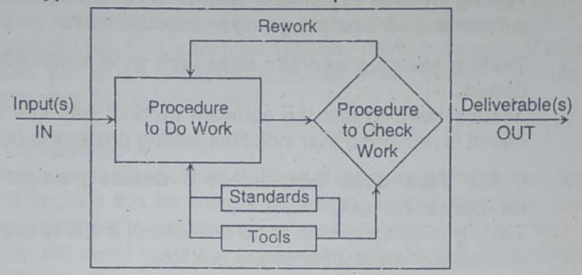
Q.12 What is a workbench?

Ans.:

- A workbench comprises some procedures defined for doing a work, and some procedures defined to check the outcome of the work.
- The work may be anything during the software development life cycle such as collecting the requirements, making designs, coding etc.
- There are standards and tools available for doing the work and checking the work in the workbench.

Q.13 What is a typical tester workbench .Explain with a suitable diagram.

Ans.:



- A tester's workbench is made of testing process, standards, guidelines and tools used for conducting tests and checking whether the test processes applied are effective or not.
- Following examples of tester's workbench.
  - Workbench for creation of test plan
  - Workbench for creation of test strategy
  - Workbench for creation of test cases
- The following is a typical workbench described for system testing execution :

  Inputs : The inputs may be test cases, test scenarios, work products, documents etc. depending on the location of workbench in the life cycle. A delivery note must be submitted along with work product to the tester mentioning any known issue before testing is done.

  Do process : The software undergoes testing as per defined test case and procedure. 'Do process' must guide the tester while doing the testing.

  Check process : The 'check process' evaluated the testing process to compare the achievements as defined in test objectives. It helps in finding whether 'Do process' have worked correctly or not.

  Output : The test report and test log are the available outputs from the testing process. The output needs to have an exit criteria definition.

Standard and Tools : Standards may include how to install the applicatio which steps needs to be followed while doing testing. Tools like defec management tools, configuration tools etc. will be available.

Rework : If 'check processes ' find that 'do processes' are not able to achieve the objectives defined for them, there must be a route of rework to be followed :
There is a rework of 'Do process' not the work product under testing.
There are two criteria's as follows :

- **Suspension Criteria :** It is to halt or suspend the testing process, if there are some major problems in process and standards or inputs.
- **Resumption Criteria :** It restarts the testing process after the suspension.

Q.14 What are the important features of testing process?

Ans.:

- **Testing is a Destructive Process, but it is constructive destruction.**
  Testing involves systematic destruction of a product with the intent to find the defects and fix it before giving to the customer.
- **Testing needs a sadistic approach with a consideration that there is a defect.**
  Testers need to test the software product with intentions to hunt defects, if defect is not found that indicates testing process is faulty.
- **If the Test does not detect a defect present in the system, It is unsuccessful test.**
  Testing process is said to be successful if it is able to find defects or threats and weaker areas of software product.
- **A test that detects defect is a valuable investment for development as well as customer, it helps in improving a product.**
  The defect finding helps in improving processes to produce a product which can result in increased customer satisfaction.
- **it is risky to develop software and not to test it before delivery.**
  Reduced or Minimal testing of the software is always risky because the software may fail at customers site.
- **With high pressure to deliver software as quickly as possible, test process, must provide maximum value in shortest timeframe.**
  Testing starts at the stage of proposal and ends only when the software application is finally accepted by the customer.
- **Highest payback comes from detecting a defect early in SDLC and preventing defect leakage/defect migration from one phase to another.**
  The investment in testing can be worthwhile only if it is capable of finding defects as soon as it is introduced in the software as well as prevent potential defects from occurring in future.
- **Organization's aim must be defect prevention defect aim must be defect prevention rather than finding and fixing.**
  The prevention mechanism is useful to prevent occurrence and recurrence of defects than finding and fixing defects.

Q.15 What are the misconceptions about testing?

Ans.: There are many misconceptions about software testing as listed below :

- **No special skills are required for testing**
  Many organizations have an approach that anyone can be put in testing, whereas for Test planning, test case writing and test data definition using different methodologies need skilled people.
- **Testers can test quality of product at the end of development process**
  This is atypical approach where system testing or acceptance testing is considered as qualification testing for software. Sometimes, the defects remain hidden for an entire life cycle of software without anybody knowing that there was a defect.
- **Defects found in Testing are blamed on Developers**
  This is another misconception where in reality two-third of defects are due to wrong requirements. Developers are just responsible for converting design into code using suitable standards. Its managements duty to give good inputs t developer's workbench.
- **Defects found by customer are blamed on testers**
  No one can say that testing can ensure 100% coverage. If no defect is found, it does not mean software is defect free. No one can blame testers for defects reported by customers.

Q.16 Define Test policy.

Ans.: Test policy is generally defined by the senior management covering all aspects of testing. It decides the framework of testing and its status in overall mission of achieving customer satisfaction. For project organizations, test policy may be defined by the client while for product organization, it is decided by the senior management.

Q.17 Define Test strategy ad Test approach and give examples of test strategy.

Ans.: Test strategy defines the action part of test policy. It defines the ways and means to achieve the test policy.

Generally, there is a single test policy at organization level for product organizations while test strategy may differ from product to product, customer to customer and time to time.

Following are some examples of test strategy may be as follows :

- Definition of coverage like requirement coverage or functional coverage etc.
- Level of testing from requirements gathering to acceptance phase.
- How much manual testing and automated testing required?
- Ratio of developers to testers.

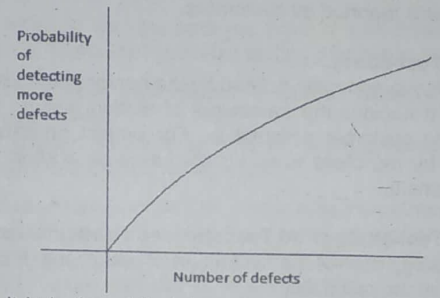
Q.18 Explain Test planning processes.

Ans.:

- Test planning involves plan for testing throughout software development life cycle. Test plan should focus on limitations and constraints of testing. Also on risks and assumptions done during testing.

- **Plan testing efforts assuming there are defects.**
  Test planning should know the number of defects it is intending to find by executing the given test plan. Test plan is successful if intended number of defects are found.
- **If Defects are not found, testing activity fails.**
  Testing is intended to find defects. If defects are not found, the testing process may be considered as defective. If defects are not found then the test cases and scenarios may be incomplete or inadequate.
- **Successful tester is the one who does not appreciates development but who finds defect in the product.**
  Success of testing is in finding a defect and not certifying that application or development process is good.
- **Testing is not a Formality to be completed at the end of development cycle.**
  Testing is a life cycle activity and should not be the last part of a development cycle before giving the application to the customer/user.

Testing Process and Number of Defects Found in Testing



- Generally , it is believed that there are fixed number of defects in a product and as testing finds more defects, chances of the customer finding the defect will reduce.
- In fact, it is reverse, there is a probability of finding some more defects.
- This is based on the principle that every application has defects and every test team has some efficiency of finding defects.
- The above diagram shows a relationship between number of defects per KLOC for the program under testing.
- The number of defects found after considerable testing will always indicate possibilities of existing number of defects.

Q.19 Explain the basic principles on which testing is based.

Ans.: The basic principles on which testing is based are given below :

- Define the expected output or result for each test case executed, to understand if expected and actual output matches or not. Mismatches may indicate possible defects. Defects may be in product or test cases or test plan.

- Developers must not test their own programs. No defects would be found in such kind of testing as approach-related defects will we difficult to find.
- Inspect the results of each test completely and carefully. It would help in root cause analysis and can be used to find weak processes. This will help in building processes rightly and improving their capability.
- Include test cases for invalid or unexpected conditions which are feasible during production. Testers need to protect the users from any unreasonable failure.
- Test the program to see if it does what is no supposed to do as well as what it is supposed to do.
- Reusability of test case is important for regression. Test cases must be used repetitively so that they remain applicable. Test data may be changed in different iterations.
- Do not plan tests assuming no errors will be found. There must be targeted number of defects for testing.
- The probability of locating more errors in any module is directly proportional to the number of errors already found in that module.

Q.20 Explain salient features of testing.

Ans.: Following are the salient features of testing :

- **Capturing user requirements :** The requirements are to be analyzed and documented by testers so that they can write the test scenario and test cases for their requirements.
- **Capturing user needs :** User needs may include present and future requirements and more importantly implied requirements.
- **Design objectives :** Design objectives indicate why a particular approach has been selected for building the software. The selection process indicates the reasons and criteria framework used for development and testing.
- **User interfaces :** User interfaces should be simple, so that the user can understand what he is supposed to do and what the system is doing. Users ability to interact with the system, receive error messages, and act according to instructions given in the user interfaces.
- **Internal Structures :** It may include reusability, nesting etc. to analyses the software product as per standards or guidelines. They are mainly guided by software designs and guidelines and standards used for designing the software.
- **Execution of Code :** Execution can only prove that application, module or program work correctly as defined in requirements and interpreted in design.

Q.21 Explain the concept of test team's defect finding efficiency.

Ans.:

- Test team efficiency is a very important aspect for development team and management. If test team is very efficient I finding defects, less iterations of testing is required.
- The test team has some level of efficiency of finding defects . suppose the application has 100 defects and the test team has an efficiency of 90 % , then it will be able to find 90 defects.
- Often, test managers and project managers try to assess the test team efficiency at some frequency. The process may be defined as below.

- Ideally, the test team efficiency may be 100 % but in reality, it may not be possible to have test teams with 100% efficiency.
- It must be very close to 100% in order to represent a good test team. As it deviates away from 100%, the test team becomes more and more unreliable.
- The development team introduces some defects in a software product and gives it to the test team. The test team completes test iterations as planned and gives the number of defects found. The development team then analyses the defects found by the test team to understand how many defects have been found by the test team , the following ratio gives test team efficiency.

  Let X = defects deliberately introduced by development.
  Y = Total defects found by testing team
  Z = Defects found by testing team but not belonging to the defects deliberately introduced by development.

  The ratio of (Y–Z)/X will give test team efficiency.

Q.22 Explain the concept of Mutation Testing. Explain test cases defect finding efficiency.

Ans.:

- Mutation testing is used to check the capability of test program and test cases to find defects. Test cases are designed and executed to find defects.
- If test cases are not capable of finding defects, it is a loss for an organization. This is also termed as test case efficiency.
- A program is written and set of test cases are designed and executed on the program. The test team may find out few defects. The original program and some defects are added deliberately. This is called 'mutant of the first program' and the process is termed 'mutation'. It is subjected to the same test case execution again. The test cases must be able to find the defects introduced deliberately in the mutant.

  Suppose, X = Defects deliberately introduced by development
  Y = Defects found by test cases in original program
  Z = Defects found by test cases in mutant.

  The ratio of (Z-Y)/X will give the test case efficiency.
- Defects found by test cases in mutant. Defects found by test cases in original program/ Defects deliberately introduced by development.

Q.23 What are the challenges faced by testers?

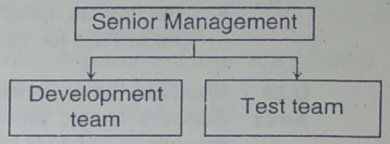
Ans.: Challenges in Testing are different on different fronts. On one front, it needs to tackle problems associated with development team, on other front, it has customers to tackle with. The major challenges faced by test teams are as follows :

- Requirements are not clear, complete, consistent, measurable and testable which may create some problems in definition of test cases and test scenarios.
- Requirements can be wrongly documented and interpreted by business analysts and system analysts, thus misunderstanding about business workflow can be a challenge.
- Testers are not able to understand the code due to lack of technical knowledge, they might not have access to code files sometimes.

- Error handing may be difficult to capture.
- Code which are not readable, maintainable, optimizable may create problems in future. Defects fixing may introduce more defects called 'regression defects'.
- Bad architecture of software cannot implement good requirement statement, creates complex code adds many defects to software product.
- Testing is considered negative activity. Often, testers need to reject builds if problems are found in it which does not satisfy exit criteria. It is difficult situation as organizational fund flow may be depending upon successful delivery of system, and it gets affected due to such rejection.

Q.24 Explain the advantages and disadvantages of independent test team approach.

Ans.: The independent test team may not be reporting to development group at all, and are independent of development activities. They may be reporting independently to senior management.



Following are the advantages and disadvantages of independent test team approach :

Advantages :

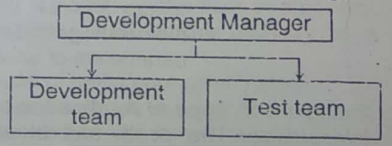
- The testers don't face delivery pressure and get enough time to perform complete testing as per definitions of coverage.
- The team does not focuses on finding a defect but they must be able to find every conceivable fault in the product before delivery.
- Since the testers and developers thinking differs , it will bring an independent view of a product.

Disadvantages :

- The team synergy can be lost as developers take pride in what they develop while testers try to break the system.
- Testers are treated as outsiders by the development team by hiding necessary product details from them.
- If the management tends towards one team the other team may feel it has no value in the organization.

Q.25 Explain the advantages and disadvantages of test team reporting to development manager approach.

Ans.: If the test team is reporting to the development manager, then they can be involve from the start of project till is finally closed.



Following are the advantages and disadvantages of test team reporting to development manager approach.

Advantages :

- Since development team and test team are a part of the same team there is better coordination between them.

- The test team gets a good understanding of requirements since they are involved in development and validation/verification activities since the very beginning.
- The testers can help improve the process by analyzing the development process.

Disadvantages :

- The testers will reply on external person rather than test manager for expert advice.
- The development manager sometimes could be more inclined towards development team.
- The testers defect finding ability reduces after they start perceiving the product like the developers.

Q.26 What are the process problems faced by testing?

Ans.: Software testing process is prone to introduce defects in the product. If the process of testing is faulty, it hives problems in terms of defects not found during testing but found by customer.

Following constituents are of processes and if faulty may cause problems :

- **People :** There is a possibility that at few instances some personal attributes and capabilities may create problem in development and testing. People may lack in skills like domain knowledge and technical skills.
- **Material :** The requirement documents, development standards, guidelines, test plans, project plan etc., may not be clear or complete or even faulty which may be responsible in introducing defects in the product.
- **Machines :** The real life scenarios created by testers with the help of simulators other software and hardware may not be able to produce full real life scenarios may be due to wrong usage of tools or machines.
- **Methods :** Methods for doing test planning, risk analysis, defining test scenarios, test cases may not be proper. These methods undergo revisions and updations as the organization matures.

Q.27 What are the cost aspect of testing?

Ans.:

- The cost of testing is the cost of the project activities in the ongoing or beyond the normal phases of development.
- It has a close relationship with the type of application being produced, methodology of development, domain and technological aspect and technical parameters of development and testing.

Type of Application and Cost of Testing :

The depth and breadth of testing has a direct relationship with size and importance of an application the user.

The testing follows Raleigh Putnam curve if the size of application is considered, as the size increases the testing efforts increase exponentially.

Cost of testing = k f(x)

Where 'k' is a constant

f(x) is a function of size of application

x is the size of the application

Development and Test methodology :
There is an impact on test efforts and costs because of development and test methodology. An organization conducting phase – wise testing has lesser cost than the organization facing testing at the last phase of development.

Waterfall model can produce robust software with lesser testing efforts. Agile methodology need huge testing because there is more stress on regression testing and so on.

Domain and Technological Aspects :
The application domain and technology plays an important role in deciding the extent of testing. If the application is critical and affects human life, there may be regulations to be followed.
The customer will be more cautious about testing and test results.
According to technological aspect object oriented development may face different challenges a compared to normal development.
Testing involves all the following three components of cost of quality.

- **Cost of prevention :** The cost of prevention is the cost incurred in preventing defects from entering into a system. It involves the following :
  - **Cost of Verification and validation :** It is the cost involved in creation of various verification and validation artifacts ,writing of test cases, test scenarios along with test data.
  - **Cost of training :** The testers may need training on the domain under testing. They may also need training on test process and various tools used for testing These are cost of training.
- **Cost of appraisal :** The first time verification / validation is considered under cost of appraisal. Test artifacts also needs reviews and testing to ensure that they are correct. The cost of reviews walkthrough, inspection, unit testing , integration testing are considered cost of appraisal.
- **Cost of failure :** The cost of failure involves retesting, regression testing and mostly cost incurred after first time validation and verification of software product. The organization must reduce the cost of failure continuously because it affects its profitability directly.

Q.28 Discuss the structured approach to testing OR Explain the four component of waste involved in testing in the last phase just before delivery of a product.

Ans.: Four components of wastes involved in this type of testing are given below :

- **Waste in wrong development :** Wrong specifications used for development, any lead to wrong development of the product which will inculcate high customer dissatisfaction, huge rework etc.
- **Waste in testing to detect defects :** If the testing focuses to find all defects in product, the cost of testing will be very high. Such kind of exhaustive testing is infeasible.
- **Wastage as wrong specifications , designs and codes :** The cost of fixing defects may be very high in the last part of testing as there are more number

of phases defect introduction and defect detection etc. Correction of the wrong specifications, designs and codes is again a costly process.

- **Wastage as system must be retested :** For every fixing of defect ,there is a possibility of some other part of software getting affected in a negative manner. One needs to test software again to ensure that fixing of software has been correct, and it has not affected the other parts in a negative manner.

Q.29 Define the terms error, bugs fault and failure.

Ans.:

- **Error :** A human action that produces an incorrect result.
- **Defect (bug, fault) :** A fault in a component or system that can cause the component or system to fail to perform its required function, e.g. an incorrect statement or data definition.
  A defect, if encountered during execution, may cause a failure of the component or system.
- **Failure :**

– A discrepancy between actual result or behavior (identified while executing test) and expected results or behavior (Identify in requirement specifications).

– Failure in a software means a software does not perform functional requirements. The failure occurs in software because of faults. Every faults or defect or bug present in the software since it was developed or changed.

– A failure has its roots in a fault in the software. These faults are also called as defects or internal errors. Programmers generaily use a term bug for the faults. Example of faults can be a communication gap between customer and developer or customer not able to specify all there requirements or customer requirements are misinterpreted or programmers might be wrongly programmed or wrong programming language selection or forgotten code in application.

– It is possible that the fault is hidden by one or more other faults present in different parts of application. In that case, a failure only occurs after masking defects that have been corrected.

– The cause of fault or defect is an error or mistake by person by environmental conditionals like radiation, magnetism, etc. that introduce hardware problems.

Comparison of mistake, error defect :

	Mistake	Error	Defect
1.	An issue identified while reviewing own documents, or peer review may be termed 'mistake'	An issue identified internally or in unit testing may be termed 'error'	An issue identified in black box testing or by customer is termed 'defect'
2.	Very low cost of finding mistakes and can be fixed immediately	Slightly more cost of finding an error and needs some time for fixing	Most costly and needs longer time for fixing defects
3.	Most of the time, problems and resolutions are not documented properly	Sometimes, problems and resolutions are documented, but may not be used for process improvements	Problems and resolutions are officially documented and used for process improvements

Q.30 Explain the process of developing test strategy.

Ans.: The process of developing test strategy goes through the following stages :

1. Select and rank Test factors for the given application.

The test team must identify critical success factors/quality factors/ test factors for the product under testing. These factors must be analysed and prioritized/ ranked.

2. Identify System Development phases and related test factors.

The importance of test factors as per system development life cycle phase must be considered to change the test approach accordingly.

3. Identify Risk associated with test factor in case not achieved.

Trade-offs may lead to few risks of development and testing the software. Customer must be involved in doing trade-offs of test factors and the possible risks of not selecting proper test factor.

4. Identify Phase in which risks of not meeting a test factor need to be addressed.

The risks may be tackled in different ways during development life cycle phases. As the phase is over, one needs to assess the actual impact of the risks and effectiveness of devised counter measures for the same.

Q.31 Explain the process of developing test methodology.

Ans.: The development of a test methodology or test plan is the job of the project level test manager. Following are the steps involved in developing of test methodology :

- **Acquire and study test strategy as defined earlier :**
  The test strategy is developed by the test team familiar with the business risks associated with software usage ,also they address the critical success factors for the project and the risk involved in not achieving it.
- **Determine the type of development project being executed :**
  The development projects can be categorized as traditionally developed software using waterfall development life cycle, COTS (commercially off the shelf purchased software which may be integrated in the given software, System maintenance software, agile development project, iterative/ prototyping project.
  Another categorization could be life affecting software, huge money affecting software, software needing simulators to test etc.
- **Determine the type of software system being made :**
  The type of software system defines how data will be processes through software, which may involve determining project scope, scope of testing changes to existing system, bug fixing etc.
- **Identify the tactical risks involved :**
  There can be structural risks(methods used to build a product), Technical risks and size risks which needs to be identified.
- **Determine when testing must occur during life cycle :**
  Testing phase may start from proposal, requirement testing till acceptance testing. The tactical test plan which will be used in execution o testing activities needs to be build.

- **Steps to develop a customized test strategy :**
  The customization of test strategy can be done by selecting and ranking quality factors, identifying system development phase where these factors must be controlled, identifying business risk and finally placing them in a matrix to analyze and take measures to control risk.

Q.32 Explain the milestones to be achieved in testing process.

Ans.: Following are few milestones commonly used by many organization :

- **Defining test policy :** Test policy at organization level defines the intent of test management.
- **Defining test strategy :** The test strategy provides the actions to the intents defined by test policy.
- **Preparing test plan :** Test plan tries to answer six basic questions like what, when, where , why , which and how etc.
- **Establishing testing objectives to be achieved :** Test objectives define the achievements that is planned, and they help in measuring the effectiveness and efficiency of a testing process.
- **Designing test scenarios and test cases :** Test scenarios are user scenarios on the basis of which the test cases are designed to test the product.
- **Writing/Reviewing test cases :** Writing and reviewing test cases along with test scenarios mostly done by senior testers or test lead.
- **Defining test data :** Test data must include valid as well as invalid set of data.
- **Creation of test bed :** Test bed defines some of the assumptions in a test plan which may induce certain risks of testing.
- **Executing test cases :** Execution of actual test cases with the test data defined for testing the software involves applying test cases as well as test data and trying to get the actual results.
- **Test Result :** Logging results of testing in test log is the last part of the testing.
- **Test result analysis :** Analyzing test results may lead to interpretation of software in terms of capabilities and weakness.
- **Regression Testing and Retesting :** Regression Testing is done to confirm that the changed part has not affected in a negative way , any other parts of software which were working earlier.
- **Root cause analysis and Corrective/Preventive actions :** Root cause analysis is required to initiate corrective action and preventive actions.

Q.33 What is Black box testing? What are its advantages and disadvantages?

Ans.: Following are most important test methodologies/approaches :

Black Box Testing :

- Black box Testing involves testing system/components considering inputs, outputs and general functionalities as defined in requirement specifications.
- It is independent of platform, database and system to make sure that the system works as per requirements defined as well as implied ones.
  It represents user scenario and actual user interactions.

Advantages :

- This is the only method to prove that software does what it is supposed to do and it does not do something which can cause a problem to user.
- Some types of testing can be done only using black box methodology like performance and security testing.

Disadvantages :

- Some logical errors in coding can be missed in black box testing as black box testing efforts are driven by requirements and not by design.
- Some redundant testing is possible as requirements may execute the same branch of code again and again.

Following are black box testing techniques :

- Equivalence partitioning
- Boundary Value analysis
- Cause and effect graph
- State transition testing
- Use case base testing
- Error guessing

Q.34 What is White box testing? What are its advantages and disadvantages?

Ans.: White Box Testing :

- White box is done on the basis of internal structures of software as defined by designs, coding standards and guidelines.
- It is a verification technique where one can ensure that software is built correctly.

Advantages :

- It can ensure that defined processes, procedures methods of development have really been followed during software testing.
- It can help in giving early warnings.

Disadvantages :

- It does not ensure that user requirements are met correctly.
- It does not establish whether decisions, conditions, paths and statements covered during reviews are sufficient or mot for given set of requirements.

Q.35 What is Gray box testing? What are its advantages and disadvantages?

Ans.: Gray Box Testing :

- Gray box testing talks about a combination of both approaches i.e white box testing and black box testing.
- It combines verification and validation techniques where one can ensure that software is build correctly and also works.

Advantages :

- It tries to combine the advantages of white box testing and black box testing.
- It requires some automation tools to conduct.

Q.36 How will testers raise awareness for testing in management?

Ans.: Following objectives help in raising management awareness for testing :

- Calculation of testing cost, effectiveness of testing and ensure that management understands the same.
- Demonstration of cost reduction and increase in effectiveness over a time span. This can be done by reducing rework and customer complaints.
- There must be awareness of need of testing activities in development as well as test team. Many developers need information about unit testing, integration testing and their role in such testing.
- Collection and distribution of testing information to all team members for improvement.
- Developing a testing budget because testing needs money, people , training and other resources etc.

Q.37 Which skills are expected by a good tester?

Ans.: Following are some of the testing skills expected by a good tester :

- **Concepts of testing :** A tester must understand methods , processes and concepts of testing. He/she must be capable of test planning, designing test scenario , writing test cases and defining test data.
- **Levels of testing :** Testers are involved in each phase of software development right from proposal and contract , followed by requirement till acceptance testing. Testers must ensure that each phase is passed successfully.
- **Techniques for Validation and Verification :** Techniques of verification/validation must be understood and facilitated by testers to the development team, customer and management.
- **Knowledge of testing standards :** Testing standards are defined by software quality team ,or also there are international standards which needs to be understood by tester and apply them effectively.
- **Risk assessment and management :** Test cases and test data must be defined by testers to minimize risks to final users in production environment.
- **Developing Test plan :** Test plan development is generally done by test managers or test leads while implementation of these plans are done by testers.
- **Defining acceptance criteria :** Testers need to define acceptance criteria for the phases and iterations of testing.
- **Execution of Test Plan :** Testers are given the responsibility of executing a test plan. It includes defining test cases, test scenarios , and their execution. They must be able to perform regression testing when planned.

❑❑❑❑❑

Pratik Dalvi

SYLLABUS

Weightage : 15 Marks

Unit Testing

Boundary Value Testing : Normal Boundary Value Testing, Robust Boundary Value Testing, Worst-Case Boundary Value Testing, Special Value Testing, Examples, Random Testing, Guidelines for Boundary Value Testing.

Equivalence Class Testing : Equivalence Classes, Traditional Equivalence Class Testing, Improved Equivalence Class Testing, Edge Testing, Guidelines and Observations.

Decision Table–Based Testing : Decision Tables, Decision Table Techniques, Cause-and-Effect Graphing, Guidelines and Observations.

Path Testing : Program Graphs, DD-Paths, Test Coverage Metrics, Basis Path Testing, Guidelines and Observations, Data Flow Testing: Define/Use Testing, Slice-Based Testing, Program Slicing Tools.

SQA - U3

Q.1 What is Unit Testing?

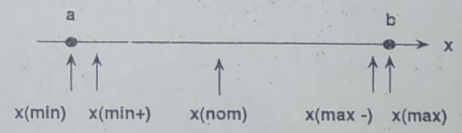
Ans.: In a procedural programming language, a unit can be :

- A single procedure.
- A function.
- A body of code that implements a single function.
- Source code that fits on one page.
- In an object-oriented programming language, there is general agreement that a class is a unit.
- The bottom line is that "unit" is probably best defined by organizations implementing code.
- The definition of a unit is a body of software that is designed, coded and tested by either one person or possibly a programmer pair.
- Unit Testing is defined as a type of software testing where individual units/ components of a software are tested.
- Unit Testing of software applications is done during the development (coding) of an application. The objective of Unit Testing is to isolate a section of code and verify its correctness. In procedural programming, a unit may be an individual function or procedure.

Q.2 What is Boundary Value Testing?

Ans.: Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.

- So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called "boundary testing".
- The basic idea in boundary value testing is to select input variable values at their:
  1. Minimum
  2. Just above the minimum
  3. A nominal value
  4. Just below the maximum
  5. Maximum



Q.3 What is Normal Boundary Value Testing?

Ans.:

- In the general application of Boundary Value Analysis can be done in a uniform manner. The basic form of implementation is to maintain all but one of the variables at their nominal (normal or average) values and allowing the remaining variable to take on its extreme values. The values used to test the extremities are:
  - Min Minimal
  - Min+ Just above Minimal
  - Nom Average
  - Max– Just below Maximum
  - Max Maximum
- The next part of boundary value analysis is based on a critical assumption; it is known as the "single fault" assumption in reliability theory. This says that

failures are only rarely the result of the simultaneous occurrence of two (or more) faults.

- Consider a function, F, of two variables x1 and x2. When the function F is implemented as a program, the input variables x1 and x2 will have some (possibly unstated) boundaries:
  $a \le x1 \le b \qquad c \le x2 \le d$

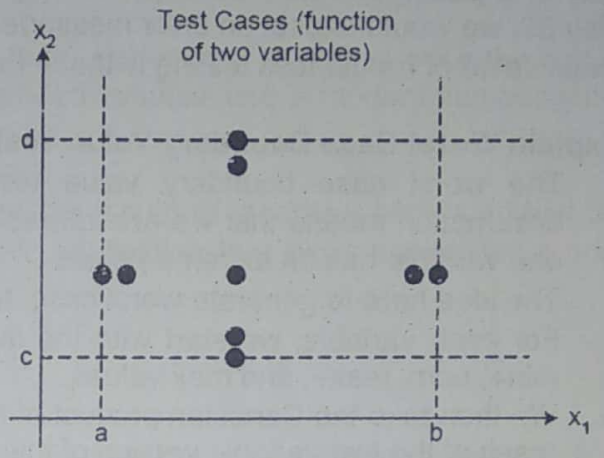


Fig.: Input domain of a function of two variables.

- Thus, the normal and robust variations cases are obtained by holding the values of all but one variable at their nominal values, and letting that variable assume its full set of test values. The normal boundary value analysis test cases for our function F of two variables.
  {<x1nom, x2min>, <x1nom, x2min+>, <x1nom, x2nom>, <x1nom, x2max–>, <x1nom, x2max>, <x1min, x2nom>, <x1min+,x2nom>, <x1max–, x2nom>, <x1max, x2nom>}

Robust Boundary Value Testing :

- Robust boundary value testing is a simple extension of normal boundary value testing.
- In addition to the five boundary value analysis values of a variable, it checks what happens when the extremes are exceeded with a value slightly greater than the maximum (max+) and a value slightly less than the minimum (min–).
- Robust boundary value test cases for our continuing example are shown in the following figure :

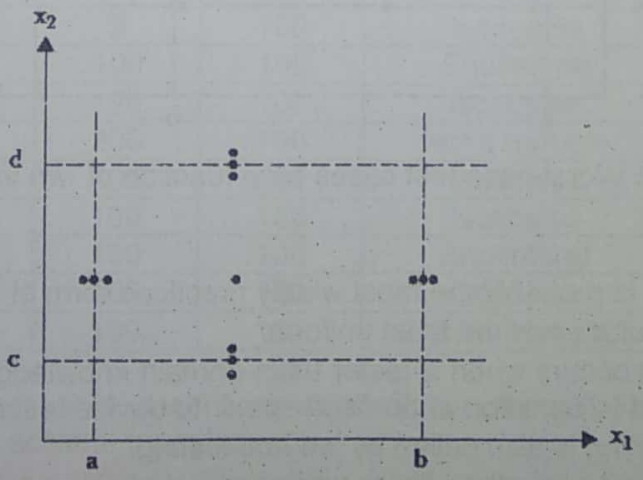


Fig.: Robustness test cases for a function of two variables.

- The most interesting part of robustness testing is not with the inputs but with the expected outputs. What happens when a physical quantity exceeds its maximum? If it is the angle of attack of an airplane wing, the aircraft might stall. If it is the load capacity of a public elevator, we hope nothing special would happen. If it is a date, like May 32, we would expect an error message.
- The main value of robustness testing is that it forces attention on exception handling.

Q.4 Explain Worst Case Boundary Value Testing.

Ans.:

- The worst case boundary value testing involves rejecting single-fault assumption means that we are interested in what happens when more than one variable has an extreme values.
- The idea here to generate worst case test cases.
- For each variable, we start with the five-element set that contains the min, min+, nom, max–, and max values.
- We then take the Cartesian product of these sets to generate test cases. The result of the two-variable version of this is shown in following figure.
- It also represents much more effort: worst-case testing for a function of n variables generates 5n test cases, as opposed to 4n + 1 test cases for boundary value analysis.
- Probably the best application for worst-case testing is where physical variables have numerous interactions, and where failure of the function is extremely costly.

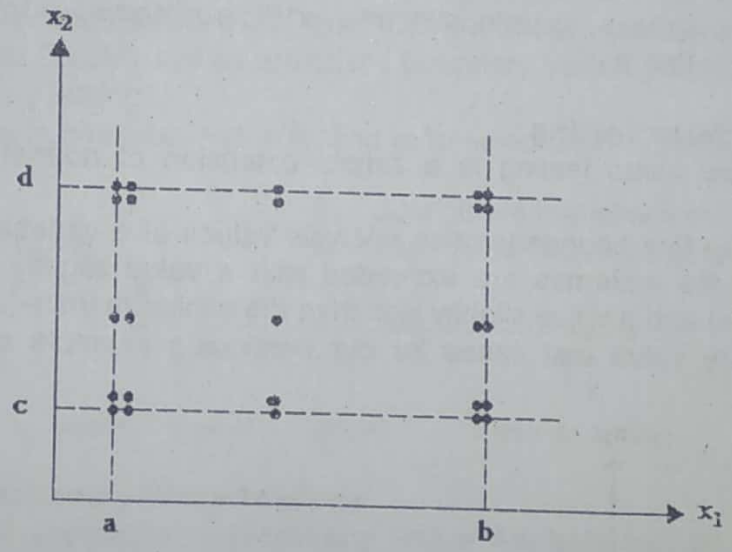


Fig.: Worst-case test cases for a function of two variables.

Special Value Testing :

- Special value testing is probably the most widely practiced form of functional testing.
- It also is the most intuitive and the least uniform.
- Special value testing occurs when a tester uses domain knowledge, experience with similar programs, and information about "soft spots" to devise test cases.
- The special value testing is also called as ad hoc testing.
- No guidelines are used other than "best engineering judgment." As a result, special value testing is very dependent on the abilities of the tester.

- Even though special value testing is highly subjective, it often results in a set of test cases that is more effective in revealing faults than the test sets generated by boundary value methods.

Random Testing :

- The basic idea of random testing is that , rather than always choose the min, min+, nom, max–, and max values of a bounded variable, use a random number generator to pick test case values.
- This avoids any form of bias in testing.
- The following Tables 1 through 2 show the results of randomly generated test cases.
- They are derived from a Visual Basic application that picks values for a bounded variable $a \leq x \leq b$ as follows:

Examples on Boundary Value Testing :

- **Triangle Problem :**

Consider a program to classify a triangle, the lower bounds of the ranges are all 1. We arbitrarily take 200 as an upper bound. The inputs are the three sides of triangle positive integers and output can be isosceles triangle, equilateral triangle or not a triangle. Design test cases using normal boundary value testing and worst case boundary value testing.

(a) Normal Boundary Value Testing :

- For each side of triangle, the test values are {1, 2, 100, 199, 200} since 1 is min, 2 is min+, 100 is nominal, 199 is max- and 200 is max.

Table 1 : Normal Boundary Value Testing Test Cases.

Case	a	b	c	Expected Output
1	100	100	1	Isosceles
2	100	100	2	Isosceles
3	100	100	100	Equilateral
4	100	100	199	Isosceles
5	100	100	200	Not a triangle
6	100	1	100	Isosceles
7	100	2	100	Isosceles
8	100	100	100	Equilateral
9	100	199	100	Isosceles
10	100	200	100	Not a triangle
11	1	100	100	Isosceles
12	2	100	100	Isosceles
13	100	100	100	Equilateral
14	199	100	100	Isosceles
15	200	100	100	Not a triangle

Notice that test cases 3, 8, and 13 are identical; two should be deleted. Further, there is no test case for scalene triangles.

(b) Worst Case– Boundary Value Testing Test Cases :

The cross-product of test values will have 125 test cases (some of which will be repeated.

The following Table only lists the first 25 worst-case boundary value test cases for the triangle problem.

Table : Worst Case -Boundary Value Testing Test Cases.

Case	a	b	c	Expected Output
1	1	1	1	Equilateral
2	1	1	2	Not a triangle
3	1	1	100	Not a triangle
4	1	1	199	Not a triangle
5	1	1	200	Not a triangle
6	1	2	1	Not a triangle
7	1	2	2	Isosceles
8	1	2	100	Not a triangle
9	1	2	199	Not a triangle
10	1	2	200	Not a triangle
11	1	100	1	Not a triangle
12	1	100	2	Not a triangle
13	1	100	100	Isosceles
14	1	100	199	Not a triangle
15	1	100	200	Not a triangle
16	1	199	1	Not a triangle
17	1	199	2	Not a triangle
18	1	199	100	Not a triangle
19	1	199	199	Isosceles
20	1	199	200	Not a triangle
21	1	200	1	Not a triangle
22	1	200	2	Not a triangle
23	1	200	100	Not a triangle
24	1	200	199	Not a triangle
25	1	200	200	Isosceles

(c) Next Date Function Problem :

Q.5 **Consider a program determining the next date with .Input day, month and year will valid ranges. As 1<= Day <= 31 , 1<= Month<= 12 and 1800<=Year<=2000. Design worst case boundary value test cases.**

Ans.: Worst case Boundary value testing test cases :

Case	Month	Day	Year	Expected Output
1	1	1	1812	1, 2, 1812
2	1	1	1813	1, 2, 1813
3	1	1	1912	1, 2, 1912
4	1	1	2011	1, 2, 2011
5	1	1	2012	1, 2, 2012
6	1	2	1812	1, 3, 1812
7	1	2	1813	1, 3, 1813
8	1	2	1912	1, 3, 1912

9	1	2	2011	1, 3, 2011
10	1	2	2012	1, 3, 2012
11	1	15	1812	1, 16, 1812
12	1	15	1813	1, 16, 1813
13	1	15	1912	1, 16, 1912
14	1	15	2011	1, 16, 2011
15	1	15	2012	1, 16, 2012
16	1	30	1812	1, 31, 1812
17	1	30	1813	1, 31, 1813
18	1	30	1912	1, 31, 1912
19	1	30	2011	1, 31, 2011
20	1	30	2012	1, 31, 2012
21	1	30	1812	2, 1, 1812
22	1	30	1813	2, 1, 1813
23	1	30	1912	2, 1, 1912
24	1	30	2011	2, 1, 2011
25	1	30	2012	2, 1, 2012

Q.6 Explain equivalence class technique.

Ans.: In equivalence class technique, the input and the output domain is divided into a finite number of equivalence classes. The use of equivalence classes as the basis for functional testing and is appropriate in situations like:

- When exhaustive testing is desired.
- When there is a strong need to avoid redundancy.
- Then, we select one representative of each class and test our program against it. It is assumed by the tester that if one representative from a class is able to detect error then why should he consider other cases. Furthermore, if this single representative test case did not detect any error then we assume that no other test case of this class can detect error. In this method we consider both valid and invalid input domains. The system is still treated as a black-box meaning that we are not bothered about its internal logic.
- The idea of equivalence class testing is to identify test cases by using one element from each equivalence class. If the equivalence classes are chosen wisely, the potential redundancy among test cases can be reduced.

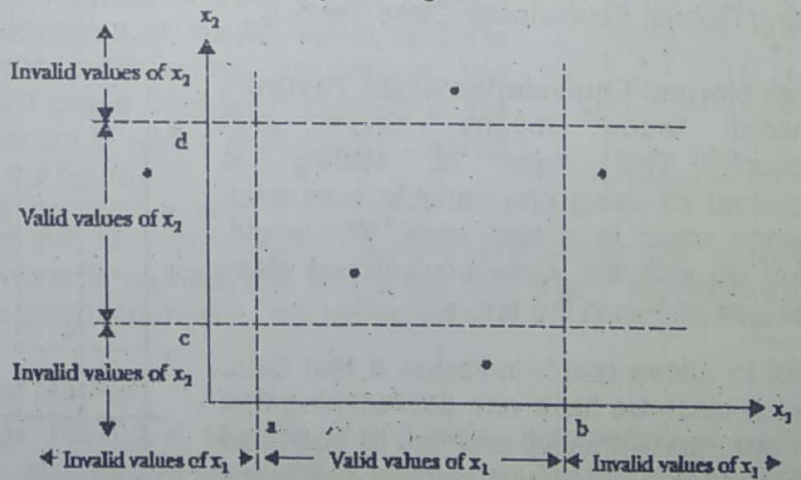


Fig.: Traditional Equivalence Class Testing.

- The traditional equivalence class testing focuses on invalid data values.
- The above diagram represents test cases for a function F of two variables x1 and x2, for n variables following steps are followed
  1. Test F for valid values of all variables.
  2. If step 1 is successful, then test F for invalid values of x1 with valid values of the remaining variables. Any failure will be due to a problem with an invalid value of x1.
  3. Repeat step 2 for the remaining variables.
- One clear advantage of this process is that it focuses on finding faults due to invalid data.

Q.7 What is improved equivalence class testing technique?

Ans.: In improved equivalence class testing, the key must be the equivalence relation selected for determining the classes .When F is implemented as a program, the input variables x1 and x2 will have the following boundaries, and intervals within the boundaries:

$a \leq x1 \leq d$, with intervals [a, b), [b, c), [c, d]
$e \leq x2 \leq g$, with intervals [e, f), [f, g]

where square brackets and parentheses denote, respectively, closed and open interval endpoints.

These ranges are equivalence classes.
Invalid values of x1 and x2 are x1 <a, x1> d, and x2 <e, x2> g.
The equivalence classes of valid values are

$V1 = \{x1: a \leq x1 < b\}$, $V2 = \{x1: b \leq x1 < c\}$, $V3 = \{x1: c \leq x1 \leq d\}$,
$V4 = \{x2: e \leq x2 < f\}$, $V5 = \{x2: f \leq x2 \leq g\}$

The equivalence classes of invalid values are

$NV1 = \{x1: x1 < a\}$, $NV2 = \{x1: d < x1\}$, $NV3 = \{x2: x2 < e\}$, $NV4 = \{x2: g < x2\}$

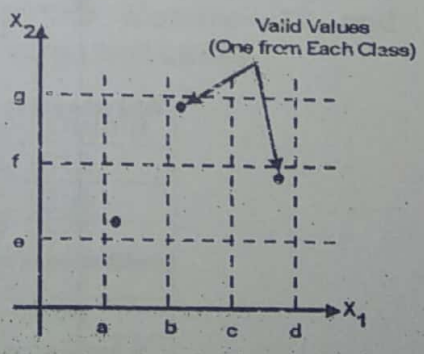
Q.8 What are the forms or types improved equivalence class testing technique?

Ans.: Following four types of equivalence class testing are presented here :

1) Weak Normal Equivalence Class Testing.
2) Strong Normal Equivalence Class Testing.
3) Weak Robust Equivalence Class Testing.
4) Strong Robust Equivalence Class Testing.

1) Weak Normal Equivalence Class Testing :

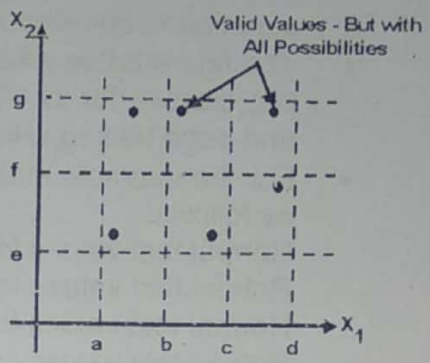
The word 'weak' means 'single fault assumption'. This type of testing is accomplished by using one variable from each equivalence class in a test case. We would, thus, end up with the weak equivalence class test cases as shown in the figure.



Each dot in above graph indicates a test data. From each class we have one dot meaning that there is one representative element of each test case. In fact, we will have, always, the same number of weak equivalence class test cases as the classes in the partition.

2) Strong Normal Equivalence Class Testing :

This type of testing is based on the multiple fault assumption theory. So, now we need test cases from each element of the Cartesian product of the equivalence classes, as shown in the figure.



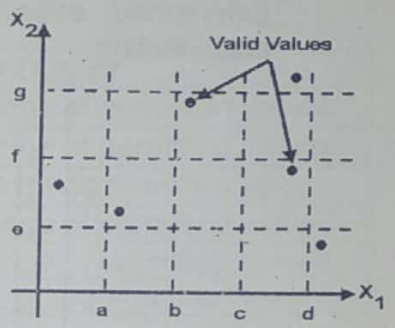
Just like we have truth tables in digital logic, we have similarities between these truth tables and our pattern of test cases. The Cartesian product guarantees that we have a notion of "completeness" in following two ways :

(a) We cover all equivalence classes

(b) We have one of each possible combination of inputs.

3) Weak Robust Equivalence Class Testing :

The name for this form of testing is counter intuitive and oxymoronic. The word' weak' means single fault assumption theory and the word 'Robust' refers to invalid values. The test cases resulting from this strategy are shown in the figure.



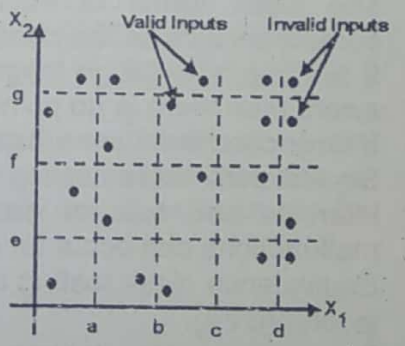
Following two problems occur with robust equivalence testing.

(a) Very often the specification does not define what the expected output for an invalid test case should be. Thus, testers spend a lot of time defining expected outputs for these cases.

(b) Strongly typed languages like Pascal, Ada, eliminate the need for the consideration of invalid inputs. Traditional equivalence testing is a product of the time when languages such as FORTRAN, C and COBOL were dominant. Thus this type of error was quite common.

4) Strong Robust Equivalence Class Testing :

This form of equivalence class testing is neither counter intuitive nor oxymoronic, but is just redundant. As explained earlier also, 'robust' means consideration of invalid values and the 'strong' means multiple fault assumption. We obtain the test cases from each element of the Cartesian product of all the equivalence classes as shown in the figure.



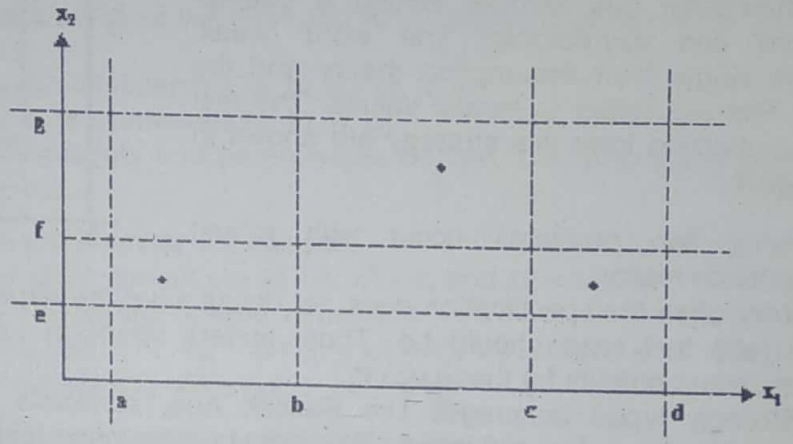
We find here that we have 8 robust (invalid) test cases and 12 strong or valid inputs. Each one is represented with a dot. So, totally we have 20 test cases (represented as 20 dots) using this technique.

Q.9 What is edge testing?

Ans.:

- The edge Testing is said to be a hybrid combination of boundary value analysis and equivalence class testing by The ISTQB Advanced Level Syllabus (ISTQB, 2012)

- The need for this occurs when contiguous ranges of a particular variable constitute equivalence classes.
- The figure below shows three equivalence classes of valid values for x1 and two classes for x2. There may be faults near the boundaries of the classes, and edge testing will help in finding these potential faults.
- For the example in the figure below, a full set of edge testing test values are as follows:
  Normal test values for x1: {a, a+, b–, b, b+, c–, c, c+, d–, d}
  Robust test values for x1: {a–, a, a+, b–, b, b+, c–, c, c+, d–, d, d+}
  Normal test values for x2: {e, e+, f–, f, f+, g–, g}
  Robust test values for x2: {e–, e, e+, f–, f, f+, g–, g, g+}
- The difference is that edge test values do not include the nominal values that we had with boundary value testing. Once the sets of edge values are determined, edge testing can follow any of the four forms of equivalence class testing.



Guidelines for Equivalence Class Testing:

The following guidelines are helpful for equivalence class testing

1. The weak forms of equivalence class testing (normal or robust) are not as comprehensive as the corresponding strong forms.
2. If the implementation language is strongly typed and invalid values cause run-time errors then there is no point in using the robust form.
3. If error conditions are a high priority, the robust forms are appropriate.
4. Equivalence class testing is approximate when input data is defined in terms of intervals and sets of discrete values. This is certainly the case when system malfunctions can occur for out-of-limit variable values.
5. Equivalence class testing is strengthened by a hybrid approach with boundary value testing (BVA).
6. Equivalence class testing is used when the program function is complex. In such cases, the complexity of the function can help identify useful equivalence classes.
7. Strong equivalence class testing makes a presumption that the variables are independent and the corresponding multiplication of test cases raises issues of redundancy. If any dependencies occur, they will often generate "error" test cases.
8. Several tries may be needed before the "right" equivalence relation is established.
9. The difference between the strong and weak forms of equivalence class testing is helpful in the distinction between progression and regression testing.

Examples :

1. Triangle problem :

Four possible outputs – NotA-Triangle, Scalene, Isosceles and Equilateral.

R1 = {<a,b,c> : the triangle with sides a.b and c is equilateral}
R2 = {<a,b,c> : the triangle with sides a,b and c is isosceles}
R3 = {<a,b,c> : the triangle with sides a,b and c is isosceles}
R4 = {<a,b,c> : sides a,b and c do not form a triangle}

Test Case	a	b	c	Expected Output
WN1	5	5	5	Equilateral
WN2	2	2	3	Isosceles
WN3	3	4	5	Scalene
WN4	4	1	2	Not a triangle

Weak robust Equivalence Class Test Cases :

Test Case	a	b	c	Expected Output
WR1	–1	5	5	Value of a is not in the range of permitted values
WR2	5	–1	5	Value of b is not in the range of permitted values
WR3	5	5	–1	Value of c is not in the range of permitted values
WR4	201	5	5	Value of a is not in the range of permitted values
WR5	5	201	5	Value of b is not in the range of permitted values
WR6	5	5	201	Value of c is not in the range of permitted values

Strong robust equivalence class test cases :

Test Case	a	b	c	Expected Output
SR1	–1	5	5	Value of a is not in the range of permitted values
SR2	5	–1	5	Value of b is not in the range of permitted values
SR3	5	5	–1	Value of c is not in the range of permitted values
SR4	–1	–1	5	Value of a,b are not in the range of permitted values
SR5	5	–1	–1	Value of b,c are not in the range of permitted values
SR6	–1	5	–1	Value of a,c are not in the range of permitted values
SR7	–1	–1	–1	Value of a,b,c are not in the range of permitted values

If we base equivalence classes on the output domain, we obtain a richer set of test cases. What are some of the possibilities for the three integers, a, b and c? They can all be equal, exactly one pair can be equal (this can happen in three ways), or none can be equal.

D1 = {<a, b, c>: a = b = c}
D2 = {<a, b, c>: a = b, a ≠ c}
D3 = {<a, b, c>: a = c, a ≠ b}
D4 = {<a, b, c>: b = c, a ≠ b}
D5 = {<a, b, c>: a ≠ b, a ≠ c, b ≠ c}

2. Next Date function Problem :

- Valid Equivalence Classes :

  M1 = {month : 1 ≤ month ≤ 12}
  D1 = {day: 1 ≤ day ≤ 31}
  Y1 = {year: 1812 ≤ year ≤ 2012}

- Invalid Equivalence Classes :
  M2 = {month : month < 1}
  M3 = {month : month > 12}
  D2 = {day: day < 1}
  D3 = {day: day > 31}
  Y2 = {year: year < 1812}
  Y3 = {year: year > 2012}

Weak Robust Test Cases :

Day	Month	Year	Expected Output
15	6	1912	16/6/1912
–1	6	1912	day not in range
32	6	1912	day not in range
15	–1	1912	month not in range
15	13	1912	month not in range
15	6	1811	year not in range
15	6	2013	year not in range

Strong robust ECT :

Test Case	Month	Day	Year	Expected Output
SR1	–1	15	1912	Value of month not in the range 1...12
SR2	6	–1	1912	Value of day not in the range 1...31
SR3	6	15	1811	Value of year not in the range 1812...2012
SR4	–1	–3	1912	Value of month not in the range 1...12 Value of day not in the range 1...31
SR5	6	–1	1811	Value of day not in the range 1...31 Value of year not in the range 1812...2012
SR6	–1	15	1811	Value of month not in the range 1...12 Value of year not in the range 1812...2012
SR7	–1	–1	1811	Value of month not in the range 1...12 Value of day not in the range 1...31 Value of year not in the range 1812...2012

➢ DECISION TABLE BASED TESTING

Q.10 Expiain Decision table based testing.

Ans.:

- Decision table ideal for describing situations in which a number of combinations of actions are taken under varying sets of conditions.
- A decision table has four portions the condition stub, the condition entries, the action stub, and the action entries.
- A column in the entry portion is a rule. Rules indicate which actions, if any, are taken for the circumstances indicated in the condition portion of the rule. In the decision table below, when conditions c1, c2, and c3 are all true, actions a1 and a2 occur. When c1 and c2 are both true and c3 is false, then actions a1 and a3 occur.

- The entry for c3 in the rule where c1 is true and c2 is false is called a "don't care" entry. The don't care entry has two major interpretations: the condition is irrelevant, or the condition does not apply.
- This structure guarantees that we consider every possible combination of condition values.
- This completeness property of a decision table guarantees a form of compiete testing.
- Decision tables in which all the conditions are binary are called Limited Entry Decision Tables (LETDs).

Table : Decision table diagram.

Stub	Rule 1	Rule 2	Rules 3, 4	Rule 5	Rule 6	Rules 7, 8
c1	T	T	T	F	F	F
c2	T	T	F	T	T	F
c3	T	F	–	T	F	–
a1	X	X		X		
a2	X				X	
a3		X		X		
a4			X			X

Example Triangle Problem :

- **Design a decision table to identify type of triangle being isosceles, scalene equilateral or not a triangle.**

c1: a, b, c form a triangle?	F	T	T	T	T	T	T	T	T
c2: a = b?	–	T	T	T	T	F	F	F	F
c3: a – c?	–	T	T	F	F	T	T	F	F
c4: b = c?	–	T	F	T	F	T	F	T	F
a1: Not a triangle	X								
a2: Scalene									X
a3: Isosceles					X		X	X	
a4: Equilateral		X							
a5: Impossible			X	X		X			

c1: a < b + c?	F	T	T	T	T	T	T	T	T	T	T
c2: b < a + c?	–	F	T	T	T	T	T	T	T	T	T
c3: c < a + b?	–	–	F	T	T	T	T	T	T	T	T
c4: b = c?	–	T	F	T	F	T	T	T	F	T	F
c4: a = b?	–	–	–	T	T	T	T	F	F	F	F
c5: a = c?	–	–	–	T	T	F	F	T	T	F	F
c6: b = c?	–	–	–	T	F	T	F	T	F	T	F
a1: Not a triangle	X	X	X								
a2: Scalene											X
a3: Isosceles							X		X	X	
a4: Equilateral				X							
a5: Impossible					X	X		X			

Following test cases can be created using the refined decision table above :

Case ID	a	b	c	Expected Output
DT1	4	1	2	Not a triangle
DT2	1	4	2	Not a triangle
DT3	1	2	4	Not a triangle
DT4	5	5	5	Equilateral
DT5	?	?	?	Impossible
DT6	?	?	?	Impossible
DT7	2	2	3	Isosceles
DT8	?	?	?	Impossible
DT9	2	3	2	Isosceles
DT10	3	2	2	Isosceles
DT11	3	4	5	Scalene

Q.11 What are advantages and disadvantages of decision table technique?

Ans.: Advantages of Decision Table Testing :

- When the system behavior is different for different input and not same for a range of inputs, both equivalent partitioning, and boundary value analysis won't help, but decision table can be used.
- The representation is simple so that it can be easily interpreted and is used for development and business as well.
- This table will help to make effective combinations and can ensure a better coverage for testing.
- Any complex business conditions can be easily turned into decision tables.
- In a case we are going for 100% coverage typically when the input combinations are low, this technique can ensure the coverage.

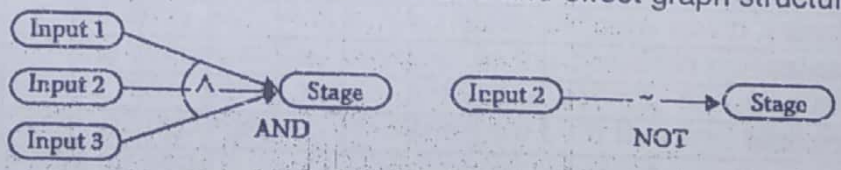
Disadvantages of Decision Table Testing :

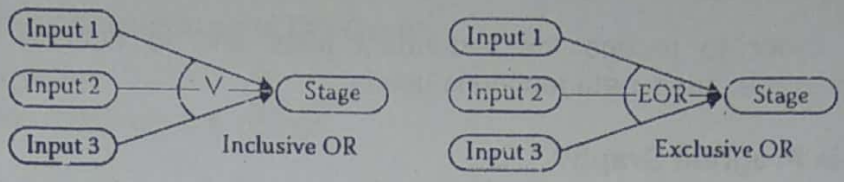
The main disadvantage is that when the number of input increases the table will become more complex.

Q.12 Explain Cause Effect Graphing Technique with an example.

Ans.:

- Cause-Effect Graphing is a technique where causes are the input conditions and effects are the results of those input conditions.
- The Cause-Effect graph technique restates the requirements specification in terms of the logical relationship between the input and output conditions. Since it is logical, it is obvious to use Boolean operators like AND, OR and NOT.
- Cause-and-effect graphs shows unit inputs on the left side of a drawing, and using AND, OR, and NOT "gates" to express the flow of data across stages of unit.
- The following Figure shows the basic cause-and-effect graph structures :





Example: Triangle problem
The causes designated by letter C are as under
C1 : Side x is less than sum of y and z
C2 : Side y is less than sum of x and z
C3 : Side z is less then sum of x and y
C4 : Side x is equai to side y
C5 : Side x is equal to side z
C6 : Side y is equal to side z

The effects designated by letter e are as under
e1 : Not a triangle
e2 : Scalene triangle
e3 : Isosceles triangle.
e4 : Equilateral triangle
e5 : Impossible

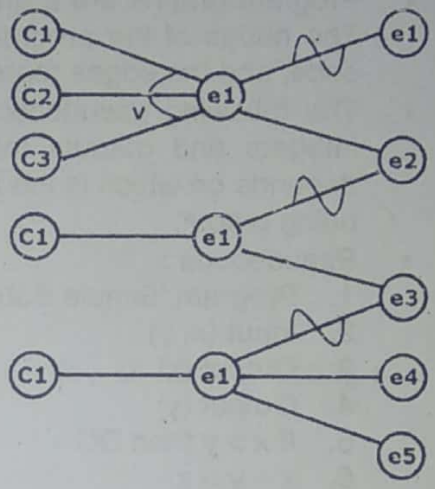


Fig.: Cause-Effect diagram for triangle

Q.13 What are the guidelines and observations from decision table technique?

Ans.:
- The decision table–based testing works well for some applications and is not worth the trouble for others.
- It works very well in the situations in which a lot of decision making takes place (such as the triangle problem), and those in which important logical relationships exist among input variables.
- The decision table technique is indicated for applications characterized by any of the following:
  (a) Prominent if–then–else logic
  (b) Logical relationships among input variables
  (c) Calculations involving subsets of the input variables
  (d) Cause-and-effect relationships between inputs and outputs
- Decision tables do not scale up very well (a limited entry table with n conditions has 2n rules).
- There are several ways to deal with this—use extended entry decision tables, algebraically simplify tables, "factor" large tables into smaller ones, and look for repeating patterns of condition entries.

Q.14 What is Path Testing?

Ans.:
- Path testing is a structural testing method that involves using the source code of a program in order to find every possible executable path.
- It helps to determine all faults lying within a piece of code.
- This method is designed to execute all or selected path through a computer program.
- Any software program includes, multiple entry and exit points. Testing each of these points is a challenging as well as time-consuming.

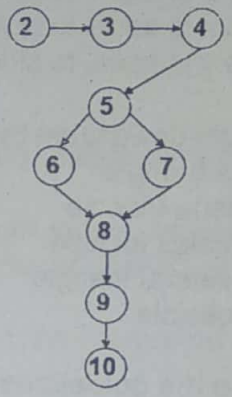
- In order to reduce the redundant tests and to achieve maximum test coverage, basis path testing is used.

Q.15 What is Program Graph?

Ans.:

- Program graphs are a graphical representation of a program's source code. The nodes of the program graph represent the statement fragments of the code, and the edges represent the program's flow of control.
- The following pseudocode for a simple program that simply subtracts two integers and outputs the result to the terminal. The number subtracted depends on which is the larger of the two; this stops a negative number from being output.
- Pseudocode :

1. Program 'Simple Subtraction'
2. Input (x, y)
3. Output (x)
4. Output (y)
5. If x > y then DO
6. x – y = z
7. Else y – x = z
8. EndIf
9. Output (z)
10. Output "End Program"



DD-Paths

- A decision-to-decision path, or DD-path, is a path of execution (usually through a flow graph representing a program, such as a flow chart) between two decisions.
- A decision-to decision path (DD-path) is the best-known form of code-based testing.
- A DD-path is a sequence of nodes in a program graph such that
  Case 1: It consists of a single node with indeg = 0.
  Case 2: It consists of a single node with outdeg = 0.
  Case 3: It consists of a single node with indeg ≥ 2 or outdeg ≥ 2.
  Case 4: It consists of a single node with indeg = 1 and outdeg = 1.
  Case 5: It is a maximal chain of length ≥ 1.
- Cases 1 and 2 establish the unique source and sink nodes of the program graph of a structured program as initial and final DD-paths.
- Case 3 deals with complex nodes; it assures that no node is contained in more than one DD-path.
- Case 4 is needed for "short branches"; it also preserves the one-fragment, one DD-path principle.
- Case 5 is the "normal case," in which a DD-path is a single entry, single-exit sequence of nodes (a chain).
- The "maximal" part of the case 5 definition is used to determine the final node of a normal (nontrivial) chain.

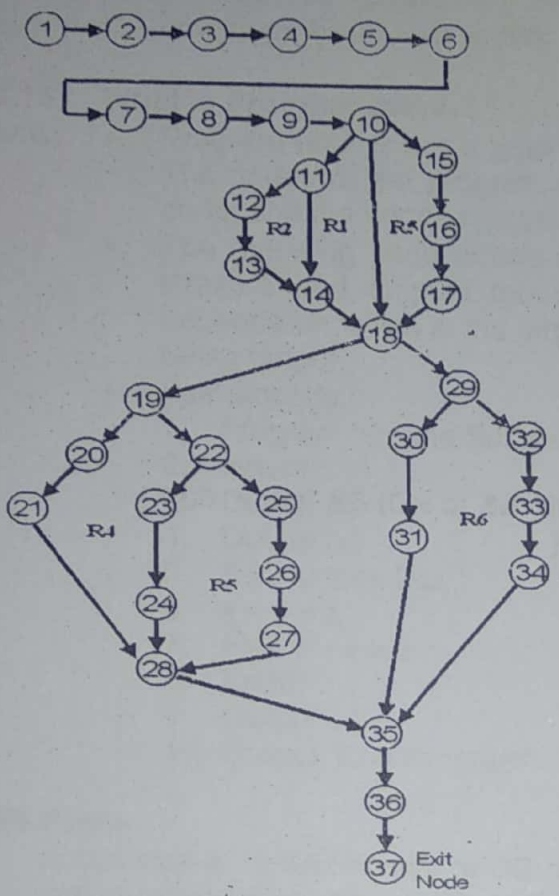
Example : Process of constructing the DD Graph

Program of triangle problem:

Step 1: Start writing the following C program

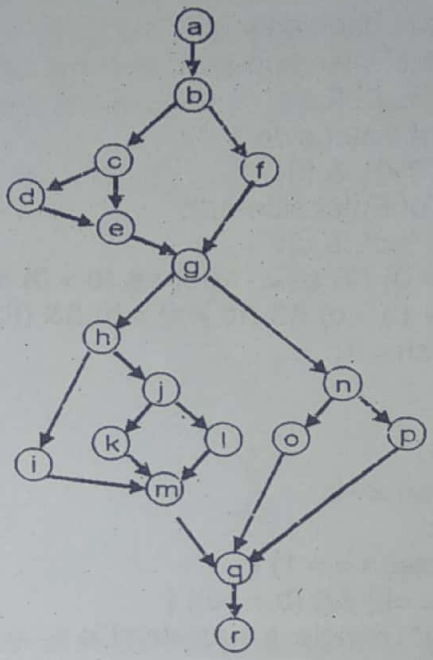
```
# include
# include
(1) int main ( )
(2) {
(3) int a, b, c, boolean = 0;
(4) printf ("nt Enter side-a :");
(5) scanf ("%d", & a);
(6) printf("nt Enter side-b :");
(7) scanf ("%d", & b);
(8) printf ("nt Enter side-c:");
(9) scanf ("%d", & c);
(10) if ((a > 0) && (a < - 100) && (b > 0) && (b < . 100) && (c > 0) && (c < =100)) {
(11) if ((a + b) > c) && ((c + a) > b) && ((b + c) > a)) {
(12) boolean = 1;
(13) }
(14) }
(15) else {
(16) boolean = -1;
(17) }
(18) if (boolean = = 1) {
(19) if ((a = =b) && (b = =c)) {
(20) printf ("Triangle is equilateral");
(21) }
(22) else if ((a = =b) I I (b = = c) I I (c = = a)) {
(23) print ("Triangle is isosceles");
(24) }
(25) else {
(26) printf("Triangle is scalene");
(27) }
(28) }
(29) else if (boolean = = 0) {
(30) printf ("Not a triangle");
(31) }
(32) else
(33) printf ("n invalid input range");
(34) }
(35) getch ( );
(36) return -1;
(37) }
```

Step 2: Draw the following Flow Graph



Step 3: Draw the following DD Path Graph

- Since, nodes 1-9 are sequential nodes in the above flow graph, hence they are merged together as a single node – "a".
- Likewise we can go on deciding the merging of nodes & arrive at the following DD Path Graph.



Test Coverage Metrics :

Test coverage metrics are a device to measure the extent to which a set of test cases covers (or exercises) a program. There are several widely accepted test coverage metrics; most of those in Table are due to the early work of E. F. Miller.

Table : Structural Test Coverage Metrics.

Metric	Description of Coverage
C0	Every statement
C1	Every DD-Path (predicate outcome)
C1p	Every predicate to each outcome
C2	C1 coverage + loop coverage
Cd	C1 coverage + every dependent pair of DD-Paths
CMCC	Multiple condition coverage
Cik	Every program path that contains up to k repetitions of a loop (usualiy k = 2)
Cstat	"Statistically significant" fraction of paths
C∞	All possible execution paths

➢ **Metric Based Testing :**

The test coverage metrics in above Table tell us what to test, but not how to test it. In this section, we take a closer look at techniques that exercise source code in terms of the metrics in above Table.

Miller's test coverage metrics are based on program graphs in which nodes are full statements, whereas our formulation allows statement fragments to be nodes.

Statement and Predicate Testing :

- Because our formulation allows statement fragments to be individual nodes, the statement and predicate levels (C0 and C1) to collapse into one consideration.
- In our triangle example (see Figure 1), nodes 8, 9, and 10 are a complete Pascal IF-THEN-ELSE statement. If we required nodes to correspond to full statements, we could execute just one of the decision alternatives and satisfy the statement coverage criterion.
- Because we allow statement fragments, it is "natural" to divide such a statement into three nodes. Doing so results in predicate outcome coverage. Whether or not our convention is followed, these coverage metrics require that we find a set of test cases such that, when executed, every node of the program graph is traversed at least once.

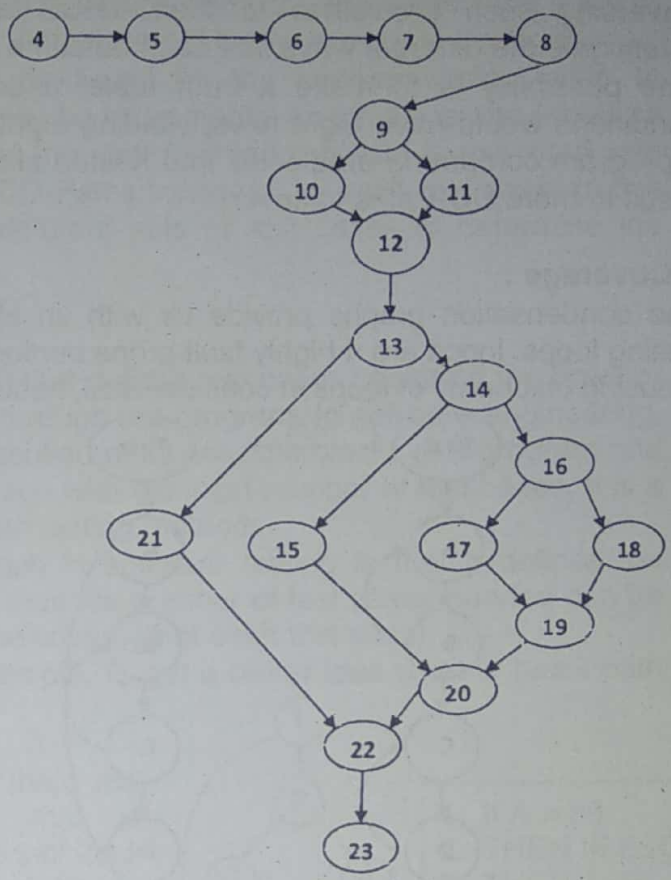


Fig. 1

DD-Path Testing :

- When every DD-Path is traversed (the C1 metric), we know that each predicate outcome has been executed; this amounts to traversing every edge in the DD-Path graph (or program graph), as opposed to just every node.
- For IF-THEN and IF-THEN-ELSE statements, this means that both the true and the false branches are covered (C1p coverage). For CASE statements, each clause is covered.

Dependent Pairs of DD-Paths :

- The Cd metric foreshadows the dataflow testing. The most common dependency among pairs of DD-Paths is the define/reference relationship, in which a variable is defined (receives a value) in one DD-Path and is referenced in another DD-Path.
- The importance of these dependencies is that they are closely related to the problem of infeasible paths.
- Example in above Figure 1, B and D are such a pair, so are DD-Paths C and L. Simple DD-Path coverage might not exercise these dependencies, thus a deeper class of faults would not be revealed.

Multiple Condition Coverage :

- Look closely at the compound conditions in DD-Paths A and E. Rather than simply traversing such predicates to their TRUE and FALSE outcomes, we should investigate the different ways that each outcome can occur.
- One possibility is to make a truth table; a compound condition of three simple conditions would have eight rows, yielding eight test cases. Another possibility is to reprogram compound predicates into nested simple IF-THEN-ELSE logic, which will result in more DD-Paths to cover.

Loop Coverage :

- The condensation graphs provide us with an elegant resolution to the problems of testing loops. loops are a highly fault prone portion of source code. To start, there is an amusing taxonomy of loops in concatenated, nested, and horrible shown in Figure 2.

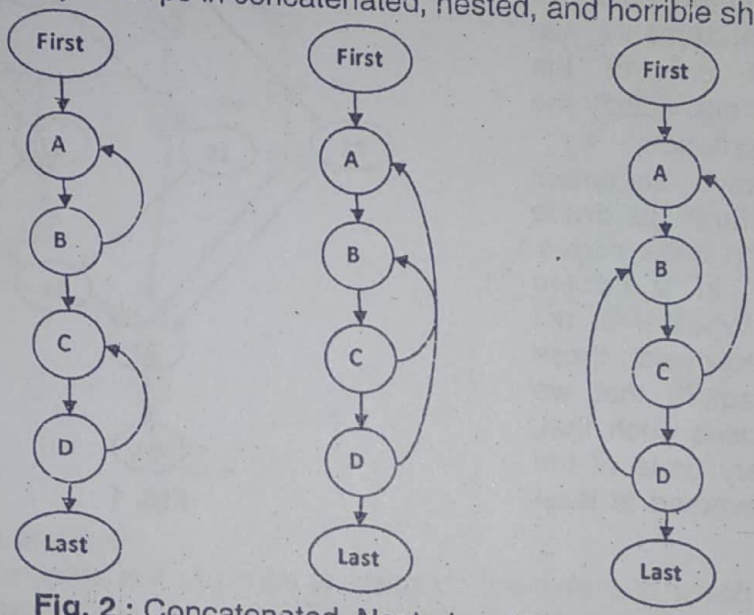


Fig. 2 : Concatenated, Nested and Knotted Loops.

- Concatenated loops are simply a sequence of disjoint loops, while nested loops are such that one is contained inside another.
- Horrible loops cannot occur when the structured programming precepts are followed. When it is possible to branch into (or out from) the middle of a loop, and these branches are internal to other loops, the result is Beizer's horrible loop. (Other sources define this as a knot—how appropriate.)
- The simple view of loop testing is that every loop involves a decision, and we need to test both outcomes of the decision: one is to traverse the loop, and the other is to exit (or not enter) the loop.
- Once a loop has been tested, the tester condenses it into a single node. If loops are nested, this process is repeated starting with the innermost loop and working outward. This result in the same multiplicity of test cases we found with boundary value analysis, which makes sense, because each loop index variable acts like an input variable.
- If loops are knotted, it will be necessary to carefully analyze them in terms of the dataflow methods. As a preview, consider the infinite loop that could occur if one loop tampers with the value of the other loop's index.

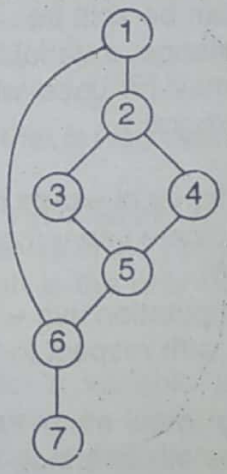
➢ **Test Coverage Analyzers :**

- Coverage analyzers are a class of test tools that offer automated support for this approach to testing management. With a coverage analyzer, the tester runs a set of test cases on a program that has been "instrumented" by the coverage analyzer.
- The analyzer then uses information produced by the instrumentation code to generate a coverage report. For example, the instrumentation identifies and labels all DD-Paths in an original program. When the instrumented program is executed with test cases, the analyzer tabulates the DD-Paths traversed by each test case. In this way, the tester can experiment with different sets of test cases to determine the coverage of each set.

Q.16 What is Basis Path Testing?

Ans.:

- The basis path testing is testing that defines test cases based on the flows or logical path that can be taken through the program. In software engineering, Basis path testing involves execution of all possible blocks in a program and achieves maximum path coverage with the least number of test cases. It is a hybrid of branch testing and path testing methods.
- The objective behind basis path in software testing is that it defines the number of independent paths, thus the number of test cases needed can be defined explicitly (maximizes the coverage of each test case).
- Here we will take a simple example, to get a better idea what is basis path testing include :
- In the example, we can see there are few conditional statements that is executed depending on what condition it suffice. Here there are 3 paths or condition that need to be tested to get the output,
  Path 1: 1,2,3,5,6, 7
  Path 2: 1,2,4,5,6, 7
  Path 3: 1, 6, 7



1. If A = 50
2. THEN IF B>C
3. THEN A=B
4. ELSE A=C
5. ENDIF
6. ENDIF
7. Point A

Q.17 What is data flow testing?

Ans.:

- Data flow testing is a form of structural testing that focus on the points at which variables receive values and the points at which these values are used (or referenced).
- Data flow testing uses the control flow graph to explore the unreasonable things that can happen to data (data flow anomalies).
- Data flow anomalies are detected based on the associations between values and variables.
  - Variables are used without being initialized.
  - Initialized variables are not used once.

Q.18 Explain define /use testing with example.

Ans.: Define/Use testing uses paths of the program graph, linked to particular nodes of the graph that relate to variables, to generate test cases. The term "Define/Use" refers to the two main aspects of a variable: it is either defined (a value is assigned to it) or used (the value assigned to the variable is used elsewhere –

maybe when defining another variable). Define/use testing was first formalised by Sandra Rapps and Elaine Weyuker in the early 1980s.

Define/use testing is meant for use with structured programs. The pro-gram is referred to as P, and its graph as G(P). The program graph has single entry and exit nodes, and there are no edges from a node to itself. The set of variables within the program is called V, and the set of all the paths within the program graph P(G) is PATHS(P).

Within the context of define/use testing, with respect to variables there are two types of nodes: defining nodes and usage nodes. The nodes are defined as follows:

- **Defining nodes, referred to as DEF(v, n):** Node n in the program graph of P is a defining node of a variable v in the set V if and only if at n, v is defined. For example, with respect to a variable x, nodes containing statements such as "input x" and "x = 2" would both be defining nodes.
- **Usage nodes, referred to as USE(v, n):** Node n in the program graph of P is a usage node of a variable v in the set V if and only if at n, v is used. For example, with respect to a variable x, nodes containing statements such as "print x" and "a = 2 + x" would both be usage nodes.

Usage nodes can be split into a number of types, depending on how the variable is used. For instance, a variable may be used when assigning a value to another variable, or it may be used when making a decision that will affect the flow of control of the program.

The two major types of usage node are:

- **P-use:** predicate use – the variable is used when making a decision (e.g. if b > 6)
- **C-use:** computation use – the variable is used in a computation (for example, b = 3 + d – with respect to the variable d).

There are also three other types of usage node, which are all, in effect, subclasses of the C-use type:

- **O-use:** output use – the value of the variable is output to the external environment (for instance, the screen or a printer).
- **L-use:** location use – the value of the variable is used, for instance, to determine which position of an array is used (e.g. a[b]).
- **I-use:** iteration use – the value of the variable is used to control the number of iterations made by a loop (for example: for (int i = 0; i <= 10; i++)).

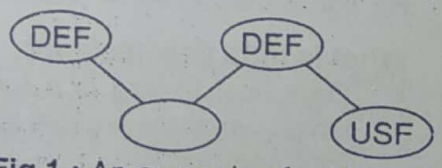


Fig.1 : An example of a DU-path.

Looking at the example above in figure 1, for the variable totalPrice table lists the defining and usage nodes (the particular instance of the variable being referred to is highlighted in bold text).

Table 1 : The defining and usage nodes for the variable totalPrice.

Node	Type	Code
4	DEF	totalPrice = 0
7	DEF	totalPrice = totalPrice + price
7	USE	totalPrice = totalPrice + price
10	USE	print("Total price: " + totalPrice)
11	USE	if(totalPrice > 15.00) then
12	USE	discount = (staffDiscount * totalPrice) + 0.50
14	USE	discount = staffDiscount * totalPrice
17	USE	finalPrice = totalPrice - discount

With these nodes, some useful paths can be generated.

- **Definition-use (du) paths:** A path in the set of all paths in P(G) is a du-path for some variable v (in the set V of all variables in the program) if and only if there exist DEF(v, m) and USE(v, n) nodes such that m is the first node of the path, and n is the last node.

  A du-path in PATHS(P)
  where the initial node of the path is the only defining node of v (in the path).

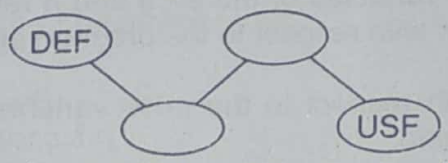


Fig.2 : An example of a DU-path that is also definition-clear.

- **Definition-clear (dc) paths:** A path in the set of all paths in P(G) is a dc-path for some variable v (in the set V of all variables in the program) if and only if it is a du-path and the initial node of the path is the only defining node of v in the path.

Figure 1 shows an example of a du-path for a variable x in a hypothetical program. However, this path is not definition-clear, as there is a second defining node within the path. Figure 2, on the other hand, is definition-clear.

Looking at the example in section 2, for the price variable there are two defining nodes and two usage nodes, as listed below:

- Defining nodes:
  - DEF(price, 5)
  - DEF(price, 8)
- Usage nodes:
  - USE(price, 6)
  - USE(price, 7)

Therefore, there are four du-paths:

- <5, 6>
- <5, 6, 7>
- <8, 9, 6>
- <8, 9, 6, 7>

All of these paths are definition-clear, so they are all dc-paths.

Q.19 Explain slice-based testing technique with example.

Ans.: The concept of program slicing was first proposed by Mark Weiser in the early 1980s. According to Weiser, "slicing is a source code transformation of a program", which allows a subset of a program, corresponding to a particular behaviour, to be looked at individually. This gives the benefit that a "programmer maintaining a large, unfamiliar program" does not have to understand "an entire system to change only a small piece". The concept of program slicing was extended to cover software maintenance by Keith Gallagher and James Lyle in 1991, extending slices to become "independent of line numbers". Amended definitions of the program slice concept are given in Paul Jorgensen's book.

A program slice with respect to a variable at a certain point in the program, is the set of program statements from which the value of the variable at that point of the program is calculated. This definition can be amended to encompass the program graph concept: by replacing the set of program statements with nodes of the program graph. This allows the tester to find the list of usage nodes from the graph, and then generate slices with them.

Program slices use the notation S(V, n), where S indicates that it is a pro-gram slice, V is the set of variables of the slice and n refers to the statement number (i.e. the node number with respect to the program graph) of the slice.

So, for example, with respect to the price variable, the following are slices for each use of the variable:

S(price, 5) = {5}
S(price, 6) = {5, 6, 8, 9}
S(price, 7) = {5, 6, 8, 9}
S(price, 8) = {8}

To generate the slice S(price, 7), the following steps were taken:

- Lines 1 to 4 have no bearing on the value of the variable at line 7 (and, for that matter, for no other variable at any point), so they are not added to the slice.
- Line 5 contains a defining node of the variable price that can affect the value at line 7, so 5 is added to the slice.
- Line 6 can affect the value of the variable as it can affect the flow of control of the program. Therefore, 6 is added to the slice.
- Line 7 is not added to the slice, as it cannot affect the value of the vari-able at line 7 in any way.
- Line 8 is added to the slice – even though it comes after line 7 in the program listing. This is because of the loop: after the first iteration of the loop, line 8 will be executed before the next execution of line 7. The program graph in figure shows this in a clear way.
- Line 9 signifies the end of the loop structure. This affects the flow of control (as shown in figure, the flow of control goes back to node 6). This indirectly affects the value of price at line 7, as the value stored in the variable will have almost certainly been changed at line 8. Therefore, 9 is added to the slice.

- No other line of the program can be executed before line 7, and so cannot affect the value of the variable at that point. Therefore, no other line is added to the slice.

The program slice, as already mentioned, allows the programmer to focus specifically on the code that is relevant to a particular variable at a certain point.

```
1   program Example()
2   var staffDiscount, totalPrice, finalPrice, discount, price
3   staffDiscount = 0.1
4   totalPrice = 0
5   input(price)
6   while(price != –1) do
7         totalPrice = total
8         Price + price input(price)
9   od
10  print("Total price: " + totalPrice)
11  if(totalPrice > 15.00) then
12        discount = (staffDiscount * totalPrice) + 0.50
13  else
14        discount = staffDiscount * totalPrice
15  fi
16  print("Discount: " + discount)
17  finalPrice = totalPrice – discount
18  print("Final price: " + finalPrice)
19  endprogram
```

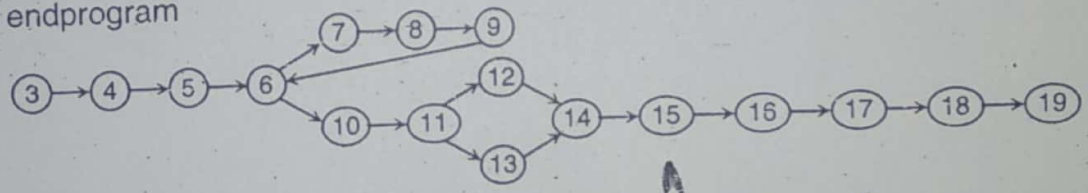


Fig. 1: The program graph for the example code.

Q.20 List different Program Slicing Tools.

Ans.; Following are a few program slicing tools; most are academic or experimental, but there are a very few commercial tools.

Following Table summarizes a few program slicing tools.

Tool/Product	Language	Static/Dynamic
Kamkar	Pascal	Dynamic
Spyder	ANSI C	Dynamic
Unravel	ANSI C	Static
CodeSonar®	C, C++	Static
Indus/Kaveri	Java	Static
JSlice	Java	Dynamic
SeeSlice	C	Dynamic

❑❑❑❑❑

Unit IV Software Verification and Validation & V-test Model

SYLLABUS **Weightage : 15 Marks**

Software Verification and Validation : Introduction, Verification, Verification Workbench, Methods of Verification, Types of reviews on the basis od Stage Phase, Entities involved in verification, Reviews in testing lifecycle, Coverage in Verification, Concerns of Verification, Validation, Validation Workbench, Levels of Validation, Coverage in Validation, Acceptance Testing, Management of Verification and Validation, Software development verification and validation activities.

V-test Model : Introduction, V-model for software, Testing during Proposal stage, Testing during requirement stage, Testing during test planning phase, Testing during design phase, Testing during coding, VV Model, Critical Roles and Responsibilities.

Q.1 What is Verification?

Ans.: The verification technique is to evaluate whether a software product fulfills the requirements or conditions imposed on them by the standards or processes.

It is also called 'static technique' as it does not involve execution of any code, program or work product. It is done by systematically reading and assessing the contents of work product with an intention of detecting defect with respect to standards or processes. It helps in identification of defect and its location which then assists in correction.

Q.2 What are advantages and disadvantages of verification?

Ans.: **Advantages :**

- The verification can confirm that the work product has followed the processes correctly as defined by an organization or customer (as the case may be).
- It can find defects – in terms of deviations from standards – easily and also, the location of the defects. This helps in fixing the defects easily as well as conducting root cause analysis so that the same defects are not repeated.
- It reduces the cost of finding and fixing defects as each work product is reviewed.
- It can locate the defect easily as the work product under review is yet to be integrated with other work products. Cost of fixing defects is less as there is no/ minimum impact on other parts of software.
- Typically peer review, superior reviews, and walkthrough can be excellent tools for training.

Disadvantages :

- It cannot show whether the developed software is correct or not. Rather, it shows whether the processes have been followed or not.
- Actual working software may not be assessed by verification as it does not cover any kind of execution of a work product. 'Fit for use' cannot be assessed in verification.

Q.3 What is a verification work bench?

Ans.: A Verification work Bench is where verification activities are conducted, and may be a physical or virtual entity. For every work bench, the following basic things are required.

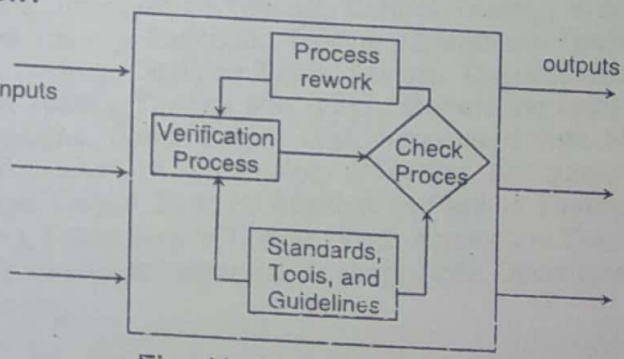


Fig.: Verification work bench

- **Inputs :** There must be some entry criteria definition when inputs are entering the work bench. This definition should match with the output criteria of the earlier work bench. For example, if we are verifying a code file, then we must have a written and compliable code as an input to the work bench.

- **Outputs :** There must be some exit criteria from work bench which should match with input criteria for the next work bench. Outputs may include review comments and the work product with defects if any.
- **Verification Process:** Verification process must describe step-by-step activities to be conducted in a work bench. It must also describe the activities done while verifying the work product under review.
- **Check Process :** Check process must describe how the verification process has been checked. It is not a verification of the work product but a verification of the process used for verification. Check process must verify that the objectives have been really achieved.
- **Standards, Tools and Guidelines :** The tools available for conducting verification activities, may be coding guidelines or testing standards available for verifications. Sometimes checklists are used for doing verification.

Q.4 Explain different methods of verification.

Ans.: Following different methods for verification of software work products :

Self-Review :

1) Self review may not be referred as an official way of review in most of the software verification descriptions, as it is assumed that everybody does a self-check before giving work product for further verification.
2) It is basically a self-learning and retrospection process.

Peer Review : Peer review is the most informal type of review where an author and a peer are involved. It is a review done by a peer and review records are maintained. A peer may be a fellow developer or tester as the case may be. There is also a possibility of superior review where peer is a supervisor with better knowledge and experience.

- **Walkthrough :** Walkthrough is a semi-formal type of review as it involves larger teams along with the author reviewing a work product.
- **Inspection (Formal Review) :** Inspection is a formal review where people external to the team may be involved as inspectors. They are 'subject matter experts' who review the work product. It is also termed 'Fagan's inspection'.
- **Audits :** Audit is a formal review based on samples. Audits are conducted by auditors who may or may not be experts in the given work product.

Q.5 What are advantages & disadvantages of self-review?

Ans.: Defects found in self-review can help in self-education and self-improvement.

Advantages :

- Self-reviews are highly flexible with respect to time and defect finding.
- Self-review is an excellent tool for self-learning, as learning from mistakes is a good way of improvement.
- There is no ego involved in self-review as the findings need not be shared with anybody.

Disadvantages :

- The approach related defects may not be found in self review as it is difficult to think in some other way.
- People involved in self-review may not conduct a review in reality due to time and focus issues.

- Something which has been implemented correctly in initial development may get changed due to personal issues.

Q.6 What are types of peer reviews?

Ans.:

- **Online Peer Review (Peer–to Peer Review) :** In such kind of review, the author and reviewer meet together and review the work product jointly. Any explanation required by the reviewer may be provided by the author.
- **Offline Peer Review (Peer Review) :** In such kind of review, the author informs the reviewer that the work product is ready for review. The reviewer may review the work product as per his time availability. The review report is send to the author along with the defects, if any. The author may decide to accept / reject the review comments or defects identified.

Advantages :

- Peer reviews are highly informal and unplanned in nature. It can happen at any time during development/ testing life cycle.
- There are no (or less) egos or pride attached between the peers, and defects are accepted easily. Peer–to–peer relationships are important in such reviews.
- Defects are discussed and decision is reached very informally. This helps in finding and fixing the defects fast.

Disadvantages :

- Peers may fix the defects without recording them, as they may not like to show defects if their partners /friends.
- The other person doing the review may or may not be an expert on the artifacts under review, and his/ her suggestions may not be valid always.
- The correct implementation may get replaced by the wrong one as reviewer may update the artifact without recording a defect. There is no discussion, and hence updation may go unnoticed.

Q.7 What are advantages and disadvantages of superior review?

Ans.: A superior such as team leader or module leader may have to review the artifacts made by his subordinates to find its suitability to find its suitability with reference to expected outcome.

Advantages :

- Approach–related defects can be found easily as superiors are expected to have a better view of the entire project. A superior may have better understanding of overall requirements and design than the author.
- Supervisors can share better ways of doing things – learned from their experience – with their subordinates. Thus, superior review is also used as a tool for training.

Disadvantages :

- Superior review can become a bottleneck if everything developed by a team needs to reviewed by the superior. For example, if there are 10 people in a team of a module leader, then reviewing their work product may hamper his own work.

- It does not work efficiently when the superior is also new to the domain / approach, or if the project is in research and development kind of environment where exact solution or even approach may not be known to anybody.

Q.8 Explain Walkthrough technique in detail.

Ans.: A walkthrough is characterized by the author of the document under review guiding the participants through the document and his or her thought processes, to achieve a common understanding and together feedback. This is especially useful if people from outside the software discipline are present, who are not used to, or cannot easily understand software development documents.

Within a walkthrough the author does most of the preparation. The participants, who are selected from different departments and backgrounds, are not required to do a detailed study of the documents in advance. Because of the way the meeting is structured, a large number of people can participate and this larger audience can bring a greater number of diverse viewpoints regarding the contents of the document being reviewed. If the audience represents a broad cross–section of skills and disciplines, it can give assurance that no major defects are 'missed' in the walk through. A walkthrough is especially useful for higher–level documents, such as requirement specifications and architectural documents.

Key characteristics of walkthroughs are :

- The meeting led by the authors; often a separate scribe is present.
- Scenarios and dry runs may be used to validate the content.
- Separate pre–meeting preparation for reviewers is optional.

Q.9 Explain Inspection technique in detail.

Ans.: Inspection is the most formal review type. The document under inspection is prepared and checked thoroughly by the reviewers before the meeting, comparing the work product with its sources and other referenced documents, and using rules and checklists. In the inspection meeting the defects found are logged and any discussion is postponed until the discussion phase. This makes the inspection meeting a very efficient meeting.

Depending on the organization and the objectives of a project, inspections can be balanced to serve a number of goals. For example, if the time to market is extremely important, the emphasis in inspections will be on efficiency. In a safety–critical market, the focus will be on effectiveness.

The generally accepted goals of inspection are to :

- help the author to improve the quality of the document under inspection;
- remove defects efficiently, as early as possible.
- improve product quality, by producing documents with a higher level of quality;
- Create a common understanding by exchanging information among the inspection participants;
- train new employees in the organization's development process;

- learn from defects found and improve processes in order to prevent recurrence of similar defects;
- sample a few pages or sections from a larger document in order to measure the typical quality of the document, leading to improved work by individuals in the future, and to process improvements.

Key characteristics of an inspection are :

- It is usually led by a trained moderator (certainly not by author).
- It is uses defined roles during the process.
- It involves peers to examine the product.
- Rules and checklist are used during the preparation phase.
- A separate preparation are carried out during which a product is examined and the defeats are found.
- The defeats found are documented in a logging list or issue log.
- A formal follow-up is carried out by the moderator applying exit criteria.
- Optionally, a causal analysis step is introduced to address process improvement issues and learn from the defeats found.
- Metrics are gathered and analyzed tc optimize the process.

Types of Review on the basis of stage/phase :
The reviews may be categorized on the basis of their happening at particular phase. During development life cycle.

➢ **In-Process Review :**
Reviews conducted while different phases of software development life cycle are going on are defined as 'in–process review'.

Milestone Review : The Milestone reviews are conducted on periodic basis depending on the completion of a particular phase or a time frame defined or a milestone achieved a software development life cycle. Three types of milestone reviews are explained below :

(i) Phase-End Review : Phase-end review is conducted at the end of a development phase under review such as requirements phase, design phase, coding and testing.

Phase–end reviews are also termed 'gate reviews' by some organizations as the gate at phase end opens only when the output criteria of the phase are achieved by the work product. Phase–end reviews are used extensively in agile development as one phase completion is marked by review before the next phase is started.

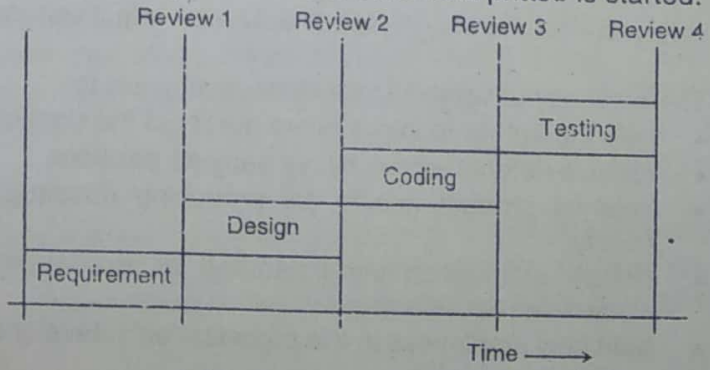


Fig.: Phase end review

(ii) Periodic Review : When one phase ends in reality another phase may be half way through. In such cases, we may have review on some periodic basis such as weekly, monthly, quarterly etc. Project plan must define various periods when review of project related activities will be conducted. Stakeholders may be required to review the progress and take actions on any impediments.

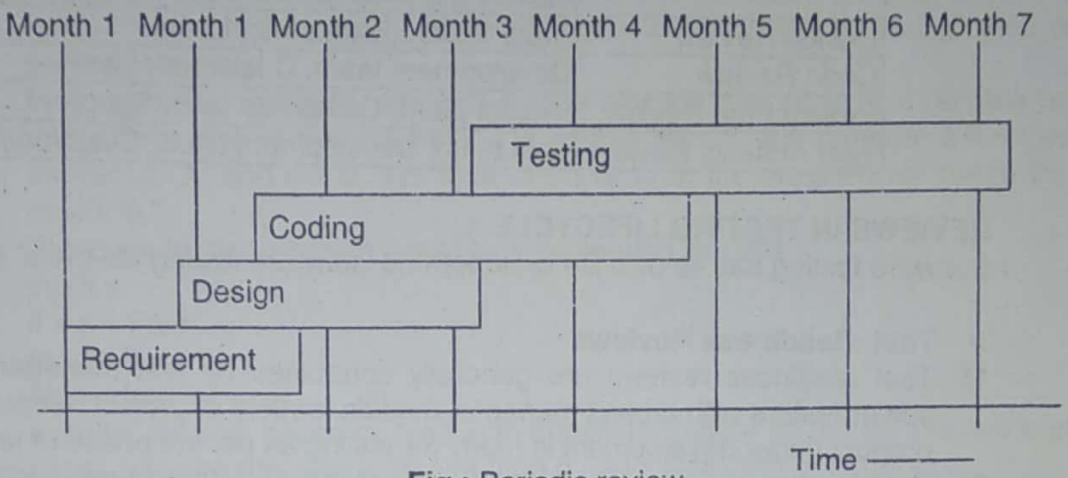


Fig.: Periodic review

Process of In–Process Review :

Following are steps of in–process reviews are given below :

- Work product, and metrics undergoing review are created and submitted to the stakeholders well in advance.
- Inputs are received from stack holders to initiate various actions.
- Risks identified by the project are reviewed and actions may be initiated as required. Risks which no longer exist may be closed and risks which cannot affect the project are retired.
- Any issue related to any stakeholder is noted and follow–up actions are initiated.
- Review reports are generated at the end of review which acts as a foundation for the next review.

➢ **Post–implementation Review :**

Post–mortem reviews/ post–implementation reviews are conducted after the project is over and delivered to the customer.

Process of Post–implementation Review :

Some of the steps commonly found in post–implementation review are as follows :

- At the end of the project when final deliverables are send to the customer and when final payments are received, post–implementation reviews are planned.
- All the stakeholders of the project gather at a predefined time.
- All the activities of project are reviewed and different stakeholders are expected to retrospect their part of work.
- Each and every activity is listed to identify if something went exceptionally good or exceptionally bad when the project was executed. All project–related metrics are reviewed.

Q.10 What List the entities involved in verification.

Ans.: Entities performing verification :

Verification	Entities Performing Verification
Requirement Review	Business analysts, System analysts, Project team including architects, developer and customer/ User
Design Review	Project team, Customer/ User
Code Review	Development team, Customer / User
Project Plan Review	Project team, Customer/ user, Suppliers
Test Artifacts Review	Test team, Development team, Customer/ User.

REVIEWS IN TESTING LIFECYCLE :

Software testing has its own life cycle termed 'software testing life cycle' (STLC).

➢ **Test Readiness Reviews**

1) Test readiness reviews are generally conducted by test managers, test leads or senior testers with project manager, module leaders or project leaders to ensure that product under development is ready for testing as per the phase of testing applicable.
2) Test readiness review is done at each stage of development such as unit testing, integration testing, interface testing, system testing, and acceptance testing.
3) **Prerequisites Testing :** Software installation has some prerequisites such as operating system, database and reporting services as the case may be. If requirement statement specifies that installation must check for the availability of such prerequisites, then it must be tested accordingly.
4) **Updations Testing :** Sometimes, it is expected that while installing software, there must be a check whether the same or similar software already exists there or not. One may experience that while installing an operating system, it checks whether some operating system already exists and what is the version of the existing system.
5) **Un-Installation Testing :** Un-installation testing is done when the developing organization wishes to check that un–installation is clean. Any file existing before installation must not get replaced while installing the application. Similarly, when an application is un–installed, all the files installed must be removed from the disk.

➢ **Test Completion Review :**

1) Test completion review is conducted when a testing cycle is completed and test results are created. Outcome of such reviews includes analysis of testing process, number and type of defects found, and number of iterations completed.
2) Test completion review must check the coverage of testing in terms of requirement coverage, functionally coverage, and feature coverage as the case may be and the number of defects found as against the targeted number of defects.
3) Final outcome of test completion is a recommendation about the status of an application.

Q.11 What is Statement Coverage?

Ans.: Statement coverage is a white box test design technique which involves execution of all the executable statements in the source code at least once. It is used to calculate and measure the number of statements in the source code which can be executed given the requirements.

Q.12 What does Path Coverage Testing mean?

Ans.:

- Path coverage testing is a specific kind of methodical, sequential testing in which each individual line of code is assessed.
- The way that path coverage testing works is that the testers must look at each individual line of code that plays a role in a module and, for complete coverage, the testers must look at each possible scenario, so that all lines of code are covered.
- In a very basic example, consider a code function that takes in a variable "x" and returns one of two results: if x is greater than 5, the program will return the result "A" and if x is less than or equal to 5, the program will return the result "B."
- The code for the program would look something like this:

```
input x
if x > 5 then
return A
else return B
```

- In order for path coverage testing to effectively "cover all paths," the two test cases must be run, with x greater than 5 and x less than or equal to 5.

Q.13 What is Decision Coverage Testing?

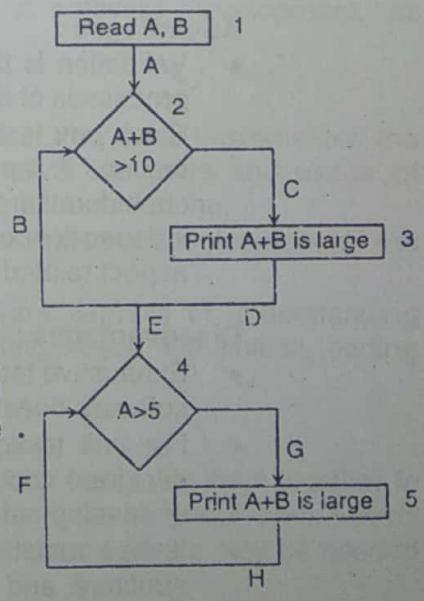
Ans.:

- Decision coverage or Branch coverage is a testing method, which aims to ensure that each one of the possible branch from each decision point is executed at least once and thereby ensuring that all reachable code is executed.
- That is, every decision is taken each way, true and false. It helps in validating all the branches in the code making sure that no branch leads to abnormal behavior of the application.

Example:

```
Read A
Read B
IF A+B > 10 THEN
    Print "A+B is Large"
ENDIF
If A > 5 THEN
    Print "A Large"
ENDIF
```

The above logic can be represented by a flowchart as:



Result : To calculate Branch Coverage, one has to find out the minimum number of paths which will ensure that all the edges are covered. In this case there is no single path which will ensure coverage of all the edges at once. The aim is to cover all possible true/false decisions.

(1) 1A-2C-3D-E-4G-5H

(2) 1A-2B-E-4F

Hence Decision or Branch Coverage is 2.

Q.14 What are concerns of verification?

Ans.: Following are concerns of verification :

- Use right verification technologies :
  Every Verification technique has some advantages and disadvantages. One may have to use verification techniques judiciously.
  For test cases and code files, peer–to–peer review may be more advantages from cost perspective while for requirement statement, inspection may be selected as a technique.
  Verification technique must be selected carefully.

- Integration of verification activities in SDLC :
  Selection of verification technique must be such that it must find the defect as early as possible. This can prevent stage contamination. The 'check' part must be very close to the 'do' part of the process so that defects in the 'do' process are found removed. This improves development process capability.

- Resources and Skills available for verification :
  The important inputs for verification are time and people with required skills. If reviewer / inspector is capable with sufficient knowledge and ability, verification can be very effective. Otherwise, it becomes ineffective as people may not find much value addition by conducting such a review. Defects found in verification must have some possibility of affecting final users, if they go undetected.

Q.15 Explain validation with advantages and disadvantages.

Ans.:

- The validation is a technique to evaluate whether the final built software product fulfills its specific intended use.
- It is meant to validate the requirements as defined in requirement specification, ensure that the application as developed matches with the requirements defined, and is fit for the intended use.
- The Validation technique is also called 'dynamic testing' as the application is executed during validation, with the intention to find defects technique.

Advantages :

- Validation is the only way to show that software developed by following all processes of development is actually functioning as desired and is usable.
- Black box testing approach is used for system, integration and acceptance testing. It represents actual user interaction with the system without any consideration of internal structures or how the system is built. It is generally independent of the platform of development, database or any other technical aspect related to software development.

Disadvantages :

- Exhaustive testing not feasible. No amount of testing can prove that software does not have defects.
- For unit testing and module testing, stubs and drivers are needed to be designed and used during testing. Stubs and drivers need additional efforts of development and testing before they can be used.
- It may result into redundant testing as the tester is not aware of internal structure, and same part of the code gets executed again and again.

VALIDATION WORK BENCH

A validation work bench is a place where validation activities are conducted on the work products, and may be a physical or virtual entity. The following basic things are required.

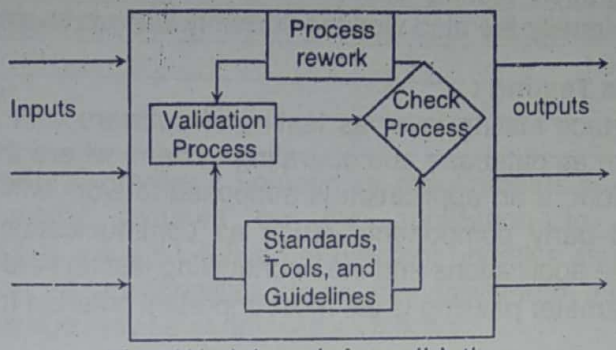


Fig.: Work bench for validation.

- **Inputs :** There must be some entry criteria definition when inputs are entering the work bench. This definition should match with the output criteria of the earlier work bench. For example, if we are validating an application or part of it, then we must have a written and compliable code as an input to the work bench along with test plan and test cases / test data as per definition of input.
- **Outputs :** There must be some exit criteria from work bench which should match with input criteria for the next work bench. Outputs may include validated work products, defects and test logs.
- **Validation Process:** Validation process must describe–step–by–step activities to be conducted in a work bench. It must also describe the activities done while validating the work product under testing.
- **Check Process :** Check process must describe how the validation process has been checked. It is not a validation of the work product but a process.
- **Standards, Tools and Guidelines :** The tools available for validation. There may be testing guidelines or testing standards available.

Q.16 What are different levels of validation?

Ans. Validation can be applied at different stages of software development, as explained below :

Unit Testing :

- Individual units need to be tested to find whether they have implemented the design correctly or not. Unit testing may involve designing and usage of stub/driver to execute units as they may not be executed alone.
- Design of stubs/ drives must be included in estimation of development for the software.
- Unit testing is done by developers as they are capable of understanding individual units under testing. Thus, unit testing needs a definition, coding and use of stub and driver for testing.

Integration Testing :

- Integration testing involves testing of many units by combining them together to form a submodule or module, as the case may be. If the integrated parts cannot create an executable on their own, designing of stubs / drivers may be needed

for their execution. In such case, integration testing is done by developers. If integration can be executed independently, it may be done by testers.

- Integration testing ensures that the units working correctly when tested individually are also working correctly when brought together.

Interface Testing :

- Interface testing involves testing of software with the environmental factors (such as database and operating system) where the application is supposed to work. If an application is supposed to work with some other applications, third–party components (such as communication software) combining all these applications and then conducting testing is termed 'interface testing'.
- Parameter passing is the most important criterion for interface testing.

System Testing :

- System testing involves end–to–end testing of a system to find the behavior of a system with respect to expectations. It needs to be done by testers as if they are the users of the system.
- System testing is actually a set of processes which may include functionality, user interface, performance, and security testing.

Cause and Effect Graphing :

- While using the system, there may be several inputs given to the system, and the system is expected to respond to those inputs.
- 'When somebody clicks a login button with valid login and password, the system allows that user to enter the system as a valid user' – this can be an example of cause and effect graphing.

Path Expression and Regular Expression :

- Path expressions can be converted into regular expressions by using mathematical laws. it can be used to examine the structural properties of a flow graph. it may include number of paths present in the code, processing time, and data flow.
  - **Path Sum :** Path sum denotes the number of paths in parallel as defined above.
  - **Path Product :** The total number of paths would be a multiple of number of path in each instance. Thus, such paths are in a series to each other.
  - **Path Loops :** Loops are infinite (or very large) number of parallel paths.

Q.17 List and explain different coverage's is validation.

Ans.: Following are some of the famous coverages offered during validation :

- Requirement coverage
- Functionality coverage
- Feature coverage

- **Requirement Coverages :** The requirements are defined in 'requirement specifications' documents. Traceability matrix starts with requirements as defined in requirement statement and goes forward upto test results. Theoretically, all the requirements as defined in requirement statement must have corresponding test cases and test results. Passing or failure of such test cases can define whether the requirements have been achieved or not.

Following table indicates requirement coverage offered in system testing :

Table : Requirement coverage definition.

Priority of requirement	Coverage offered
P_1 (Must)	100%
P_2 (Should be)	50%
P_3 (Could be)	25%

- **Functionality Coverage :** Sometimes, requirements are expressed in terms of functionality required for the application to work successfully. Functionalities are also defined at different priorities as per their importance to the users. All functionalities are not mandatory from the user's perspective. Thus, coverage may be expressed as follows :

Table : Indicates functionality coverage offered in system testing.

Priority of functionality	Coverage offered
P_1 (Must)	100%
P_2 (Should be)	50%
P_3 (Could be)	25%

Functionality coverage cannot help in identifying the coverage established by non–functional requirements such as performance and user interface.

- **Feature Coverage :**

1) Feature coverage talks about covering a feature required by the user. it may mean covering atleast one of the functionalities which represents a feature provided, even if there are multiple ways of doing things.
2) An application may need some features as defined in requirements. Features are a group of functionalities doing same/ similar things. The same features may have different ways of doing things, and of them is a functionality for the application. Accomplishing atleast one of the functionalities may help in achieving the user requirements as a feature.

Thus coverage may be expressed as follows :

Table : Features coverage definition.

Priority of feature	Coverage offered
P_1 (Must)	100%
P_2 (Should be)	50%
P_3 (Could be)	25%

Q.18 Explain the levels of acceptance testing.

Ans.: The acceptance testing is generally done by the users and/ or customers to understand whether the software satisfies their requirements or not, and to ascertain whether it is fit for use. There are two (sometimes three) levels of acceptance testing.

- **Alpha Testing :** Alfa testing represents the testing done by the customer in development environment in front of the development team. The testing is done at development site with dummy data, either created by developers or shared by customer. In reality, alpha testing may be done by testers in front of the customer to show that software is working. It can be used as a tool for training key users on the application. Any major problems in terms of

requirement understanding, functioning of software, etc. can be cleared during alpha testing before software is formally delivered. This is also termed as 'dry run' by some people.

- **Beta Testing :** Beta testing represents a business pilot where testing is actually conducted by customer in production / semi – production environment. Tester/ developers may be appointed to help the customer in testing the application, and recording the problems, if any. Beta testing is extensively used as parallel testing to give confidence to the users that the software really works as per established existing system at customer's place.
- **Gamma Testing :** Gamma testing is used for limited liability testing at selected places. Gamma testing is generally done by the product development organization that wishes to have limited market demonstration and piloting of an application in the market. The application is given to few people for using it in production, environment, and feedback is obtained from them about the features incorporated as well as something missing or wrongly interpreted.

Q.19 Explain software development verification and validation activities.

Ans.: The verification and validation activities are spread over the life cycle of software development. The cycle of verification and validation may include the following :

1) Conceptualization : Conceptualization means converting the thoughts or concepts into reality. It may be through proof of concept approach or 'prototyping' approach.
   In Conceptualization, the main stress of verification and validation activity is to determine feasibility of an approach for the project. Feasibility may be depending upon various factors such as technical feasibility, economic feasibility, and skill availability.
2) Requirement Analysis : Requirement analysis verification and validation shows the feasibility of requirements and gives inputs to design approach. It can help in finding the gaps between proposal and requirements so that further clarifications can be achieved.
3) Design : Verification and validation of design include understanding that design is complete in all respects and matches with requirements. Design validation involves dummy execution of a product design through dataflow diagrams or prototyping, to find whether the design reflects the requirements correctly along with its feasibility.
4) Coding : coding involves code review and testing of the units, as part of verification and validation to make sure that requirements and design are correctly implemented.
5) Integration : Integration validation and verification show that the individually tested units work correctly when brought together to form a module or submodule. It involves testing of modules / submodules to ensure proper working with respect to requirements and designs.
6) Testing : Test artifacts such as test plan, test scenario, test bed, test cases, and test data must be subjected to verification and validation activities.
7) Installation : Application must be tested for installation, if installation is required by the customer. The documentation giving instructions for

installation must be complete and sufficient to install the application. Verification and validation must ensure that there is adequate support and help available to common users for installation of an application.

8) Documentation : There are many documents given along with a software product such as installation guide and user manual. The documents must undergo validation to ensure that documents and product are in synchronize with each other.

V–TEST MODEL

Q.20 Explain V–Test Model.

Ans.: 1) The V Model stands for Validation Model.

2) Validation model describes the validation activities associated with different phases of software development.

Following are activities done at various phases :

- At the requirement phase, there is system testing and acceptance testing, where system testers and users confirm that requirements have been really met or not.

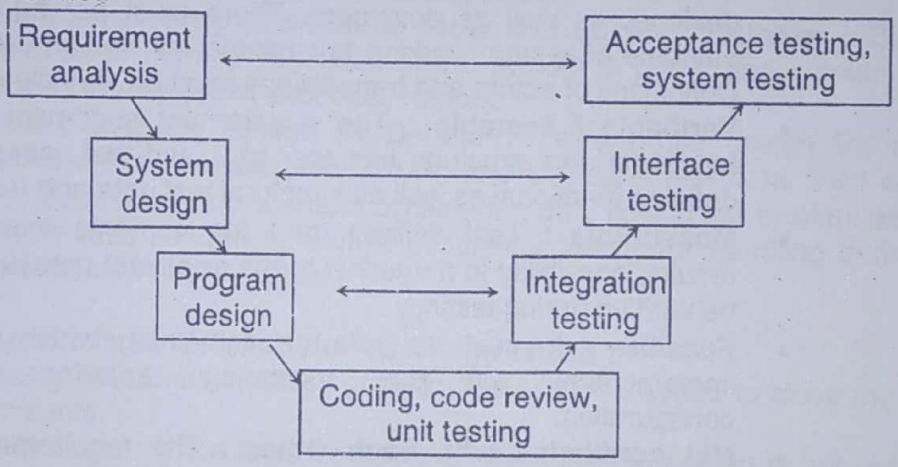


Fig.: V model for testing (validation model)

- The Design phase covers design specification testing as well as structural testing.
- Program–level designs are associated with integration testing.
- At code level, unit testing is done to validate individual units, in the program.

Q.21 Explain testing during proposal stage.

Ans.:

- The making of Requirement statement from system proposal.
- A proposal is created when the customer asks for information, quotation, proposal, etc.
- At proposal stage, system description may not be very clear. It must talk about major problems to be solved, the possible solution for which the system is designed, and any major constraints in such definition.
- People use different approaches such as formal/ informal proof of concept, modeling prototyping, etc.
- The success of all these approaches during proposal stage lies in successfully defining the problem and proposed solution.

- The feasibility study may or may not be a part of the proposal.
- Feasibility study is done by the customer as well as the supplier in search of a possible solution/ approach to solve the problem faced by the customer.
- It may include technical feasibility, economic feasibility, implementation feasibility, organizational fit, process fit, and people fit.

Q.22 Explain Testing during requirement stage.

Ans.: The Requirement gathering stage focuses on covering all the requirements for the system, defined in different groups such as technical, economical, legal, operational and system requirements.

Following are characteristics of good requirements :

- **Adequate :** The requirements must explain the entire system covering end–to–end scenario from the user's side. Requirements must not talk about any impossible thing happening.
- **Clear/Unambiguous :** The customer expectation must be reflected in requirements clearly. Requirements must be very clear to the architect, designer as well as developer. There must not be any doubt about the outcome of system working to customer as well as developing organization. Definitions of actors and transactions must be very clear.
- **Verifiable / Testable :** The requirement statement is used for defining functional and structural test scenarios and test cases. It must be used for defining functional as well as structural test data and test events.
- **Measurable :** Test defined from requirements must have the expected results, possibility in numerical terms or atleast measurable terms which can be verified during testing.
- **Feasible :** It must not refer to something which is impossible, or not implementable with given technology, approach, software or system configuration.
- **Not conflicting with Each Other :** The requirements must not specify anything which is contrary to each other/ Requirements must be supplementing and supporting each other. In case of any conflicting requirements, one must ask the customer for a tradeoff decision.

Q.23 Explain testing during test-planning phase.

Ans.: The Verification of testing artifacts may include the following, during test planning phase :

- **To Generate Test Plan to support Development Activities :** It must describe respective verification and validation activities to be conducted during each phase of software development life cycle. It must contain methods used for defining test data, test cases, test scenario, and execution of testing activities.
- **To Generate Test Cases Based System Structure :** It must be used to define test data using different techniques available such as boundary value analysis, equivalence partitioning, error guessing, and state transition. Test cases and test data must be derived from test scenario.
- **To analyse Requirement / Design Coverage :** It is very difficult for any test team to create a test suite to cover 100% requirements and designs

produced during software development life cycle. The test manager decides the objective of requirement coverage and design coverage in test plan, and the customer approves it. Coverage less than 100% indicates a risk, and customer must be involved in making such decision.

Testing during Design Phase :

Verification and validation of design may include the following :

- **Consistency with respect to Requirements :** The designs must be consistent with requirements defined. If there is any trade–off between the requirements for implementation, it must be mentioned clearly in design, and customer approval should be taken.
- **Analyse Design for Errors :** Errors in the design will directly reflect the errors in coding and application so developed. Consistency between design elements must be maintained to avoid any mismatches.
- **Analyse Error Handling :** Error handling of all kinds and possibilities must be covered in designing aspects. An organization must have standards for error handling, error messaging and user interactions so that designers are very clear about handling them during design.
- **Developers Verify information Flow and Logical Structure :** Data flow diagrams or dummy execution of the system is done, and outputs derived are measured against expected outputs.
- **Testers Inspect Design in Details :** Testers use design for defining structural test scenario, test cases, and structural test data. Test scenario must be end–to–end scenario considering data flow in the system, and must contain valid as well as invalid conditions, and error handling during user interactions with the system.

The Aspects to be checked during design phase :

1) Missing Test Cases : Missing test cases must be added to close the review comments.
2) Faulty Logic : If the logic or algorithm described in design is not correct as per requirement statement, then it must be found by testing.
3) Module Interface Mismatch : The data input / output from one module to another must be checked for consistency with design.
4) Data structure Inconsistency : Review of mismatch between data structures and definitions between different modules and system must be done for verification of designs.
5) Erroneous Input/ Output : If the system needs to be protected from erroneous operations such as huge / invalid input / output. The design must describe how it will handle the situation.
6) User Interface Inadequacies : The user interfaces defined by the design must be adequate for the purpose of communication, so that users can work with the system comfortably.
7) Correctness of Decisions and Conditions along All Paths : A system may go through different paths and branches of algorithms based on situations faced by it. The design must be such that all the possible conditions must be defined completely.
8) Inconsistency with respect to Requirements and High–Level Design : Design and development must be consistent with requirements and high–level design.

Q.24 Explain what aspects to be checked during coding.

Ans.:

- **Coding Standards / Guidelines Implementation :** When the code files are written, the developers must use these standards / guidelines. While reviewing code, the peer must make sure that standards and guidelines are implemented correctly. Coding the standards and guidelines define the best way or suggested way of doing things. It helps in optimization and better readability/ maintainability of code in future.
- **Coding Optimization :** The coding standards must also talk about optimization of code. It talks about how nesting must be done, how declaration of variables and functions must be done, how reusable components must be used and so on. Peer review must verify that the written code is properly adhering to optimization guidelines.
- **Code Interpreting Design :** Coding must interpret designs correctly. Coding files and what they are supposed to implement must be defined in low–level design. There must not be anything more or less than what has been defined in low–level design.
- **Unit Testing :** Unit testing must be done by the developers to ensure that written code is working as expected. Sometimes, unit testing is done by peer of an author (of a code) to maintain independence of testing with respect to development. Unit test cases must cover valid as well as invalid conditions faced by users. Unit test case logs must be prepared and available for peer review.

VV Model :

1. 'VV Model' talks about verification and validation activities associated with software development during entire life cycle.
2. It is also termed 'Verification and Validation Model' or 'Quality model'.
3. 'VV Model' considers all the activities related to verification as well as validation.

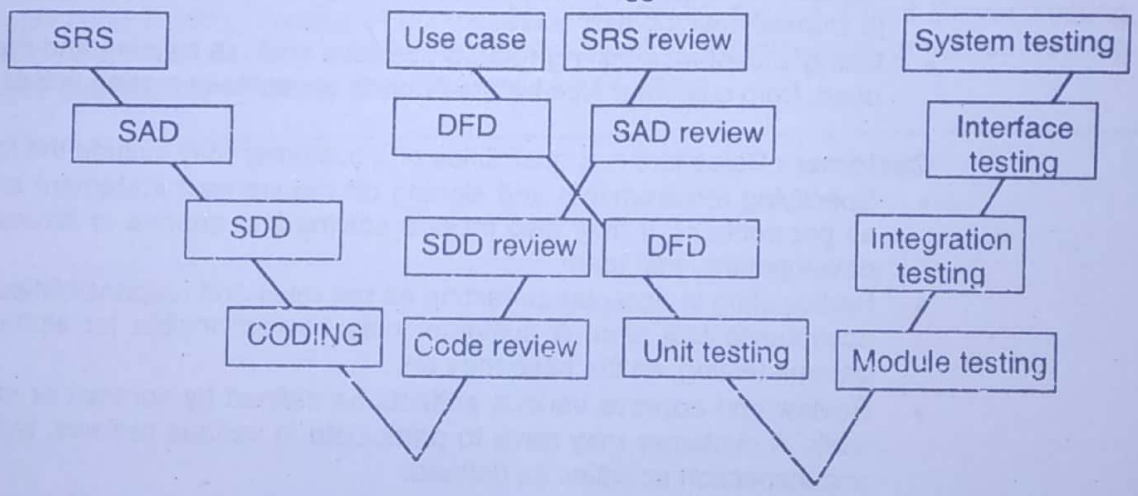
- **Requirements:** Requirement review may concentrate on the structure of requirement specifications statement using a template and content of requirement specifications as per process definition.
  1) **Requirement Verification :** Generally requirement verification is done through inspection of requirement specification document using checklist, standards or guidelines. Experts in domain, customer representative and other stakeholders may be involved in conducting such an inspection.
  2) **Requirement Validation :** Requirement validation takes place at two or more stages during software development. The first stage of validation involves writing complete use cases by referring to requirement statement. Any assumption made while writing use case may be a possible gap unless all stakeholders agree that they are implied requirements. Use cases must cover all the possible scenarios. The second stage of validation is through system testing. System test scenario and test cases are defined using requirement specifications. Requirement traceability through test scenario and test cases establish coverage of requirements.
     Requirement validation also happens when customer conducts acceptance testing on the work product.

- **Design :**
  1) **Design Verification :** Verification of design may be a walkthrough of design document by design experts, team members and stakeholders of the project.

Traceability of design with requirement must be established in requirement traceability matrix. Many organizations follow some specific tools or methodologies (like UML) to create designs. Use of any case tool must be validated so that defects introduced by tools are known beforehand.

2) **Design Validation :** Validation of design can happen at two or more stages during software development life cycle. The first stage of validation happens when data flow diagrams can be created by referring to the design document. If the flow of data is complete, design is considered to be complete. The second stage of validation happens at integration testing and interface testing. Integration testing is an activity to bring the units together and test them as a module. All units are created as per low–level design and tested in unit testing. Interface testing is an activity of testing connectivity and communication of application with the outside world. Once the system is integrated as defined in earlier phase, one may have to test it for outside connections like database, browser, operating system, communication software and user interface. Interface is defined by architectural designs.
Another design validation, though by review, happens when a customer reviews the design specification and signs off as accepted.

- **Coding :**
  1) **Code Verification :** Peer review helps in identification of errors with respect to coding standards, indenting, standards, commenting standards, and variable declaration issues. Checklist approach is used in code review.
  2) **Code Validation :** Validation of coding happens through unit testing where individual units are tested separately. The developer may have to write special programs (such as stubs and drivers) so that individual units can be tested. The executable is created by combining stubs, drivers and the units under testing. This executable is tested with the test cases mainly derived from low–level design. Test results and defects are then logged.



Q.25 What are / identify critical roles and responsibilities?

Ans.: Following are the roles & responsibilities according to phase :

Development : Some of the activities related to development may be as follows:

1. Project planning is the foundation of the project's success.

2. Project planning activities requirement elicitation, estimation, project planning, scheduling and definition of quality attributes required by the customer.
3. Resourcing may include identification and organization of adequate number of people, machines, hardware, software and tools as required by the project.
4. Interacting with customer and other stakeholders as per project requirements.
5. Defining policies and procedures for creating development work, verification and validation activities to ensure that quality is built properly, and delivering it to test team/ customer as the case may be.
6. Supporting testing team by acknowledging defects, giving inputs about requirements, giving executable on time, solving the queries raised by testing team or sending it to customer, as the case may be.

Testing : Roles and responsibilities of test team may include the following :

- Test planning including test strategy definition, and test case writing. Test planning may include estimation of efforts and resources required for testing.
- Resourcing may include identification and organization of adequate number of people, machines, hardware, software, and tools as required by the project. It may also involve an assessment of skills required by project, skills already available with them and any training needs.
- Interacting with customer and other stakeholders as per project requirements. It may include asking queries, responding to customer/ development team requests and so on.
- Defining policies and procedures for creating and executing tests as peer test strategy and test plan. Testers may have to take part in verification and validation activities related to test artifacts.
- Supporting development team by providing adequate information about the defects. If required, testers may have to reproduce defects in front of customer/ development team.
- Doing acceptance testing related activities such as training and mentoring to users from customer side before / during acceptance testing activities.

Customer : Roles and responsibilities of a customer may include the following :

- Specifying requirements and signing off requirement statement and designs as per contract. It may also include solving any queries or issues raised by development / test team.
- Participating in acceptance testing as per roles and responsibilities defined in acceptance test plan. A customer may be responsible for alpha, beta and gamma testing, as the case may be.
- Review and approve various artifacts as defined by contract or statement of work. A customer may have to participate in various reviews, walkthroughs, and inspection activities as defined.

❑❑❑❑❑

Unit V

Levels of Testing & Special Tests

SYLLABUS **Weightage : 15 Marks**

Levels of Testing : Introduction, Proposal Testing, Requirement Testing, Design Testing, Code Review, Unit Testing, Module Testing, Integration Testing, Big-Bang Testing, Sandwich Testing, Critical Path First, Sub System Testing, System Testing, Testing Stages.

Special Tests : Introduction, GUI testing, Compatibility Testing, Security Testing, Performance Testing, Volume Testing, Stress Testing, Recovery Testing, Installation Testing, Requirement Testing, Regression Testing, Error Handling Testing, Manual Support Testing, Intersystem Testing, Control Testing, Smoke Testing, Adhoc Testing, Parallel Testing, Execution Testing, Operations Testing, Compliance Testing, Usability Testing, Decision Table Testing, Documentation Testing, Training testing, Rapid Testing, Control flow graph, Generating tests on the basis of Combinatorial Designs, State Graph, Risk Associated with New Technologies, Process maturity level of Technology, Testing Adequacy of Control in New technology usage, Object Oriented Application Testing, Testing of Internal Controls, COTS Testing, Client Server Testing, Web Application Testing, Mobile Application Testing, eBusiness eCommerce Testing, Agile Development Testing, Data Warehousing Testing.

Q.1 Describe the proposal review process.
OR
Explain proposal testing.

Ans.:
- On the basis of Request for Proposal(RPF) or Request for Information (RFI) or Request for Quotation(RFQ) a proposal is made to the customer.
- A proposal made in response to such request is reviewed by an organization using following techniques before sending it to the customer.
- Technical Review :
  - It involves checking the technical feasibility of the system such as availability and requirement of skill sets, hardware / software, time and efforts required for development.
  - It described the overall approach technically required to develop the product.
- Commercial Review :
  - It involves the financial feasibility and other types of feasibilities involved with respect to the business.
  - It includes gross margins of the project, total cost, fund flow etc.
- Validation of proposal : The proposal involves making of a prototype or proof of concept to explain the proposed approach to the problem of the customer. The validation of the proposal can be made on the basis of such prototypes.

Q.2 Describe requirement verification or validation process.
OR
Explain Requirement Testing.

Ans.:
- The requirement creation required requirement gathering from the customer and arranging them to verify and validate them.
- The requirements can be categorised as technical, economical, legal, system, operational, etc., also as stated/implied, generic/specific and present/future.
- The requirement testing ensures that the requirements satisfy the following conditions.
  - **Clarity :** The requirements must be stated very clearly and also the expectations from it must be clear
  - **Complete :** The requirement statement must be complete and include all business aspects of the system also any assumption must be documented and approved by the customer.
  - **Measurable :** The requirements must be measurable or quantifiable in numerical terms. Qualitative terms like high, good must be avoided because there can be various definitions of such terms.
  - **Testable :** The requirement must be testable by creation of use cases or scenarios which can be used to test the system.
  - **Non Conflicting :** The requirements must not conflict or contradict with each other. The conflicting requirements indicates problems in requirement gathering process.
  - **Identifiable :** Each requirement must be identifiable distinctively from the other requirements .It can be done using requirement traceability matrix.
- **Validation of Requirements :** In case the complete creation of use cases can be done using the requirement statements and it justifies the working of the system then the requirements are said to be valid.

Q.3 Describe design verification or validation process.
OR
Explain Design Testing.

Ans.:
- The low level architectural designs are made by the system architect based on the requirement statements also it must be traceable to the requirements.
- The design must possess the following characteristics.
  - **Clarity :** The design must cover all the inputs, outputs, interfaces, functions, tables, procedures components etc.
  - **Complete :** The design must be complete in all the aspects .It must define the parameter passing and receiving ,complete formats of the data being handled etc.
  - **Traceable :** The design must be traceable back to the requirements, any requirement without the corresponding design must be checked for.
  - **Implementable :** The design must introduce the communication between different components of the system, also it must be easily implementable in the code and developed by the selected technology by the development team
  - **Testable :** The design must help the testers in making structural test cases.
- **Validation of Design :** The design testing shows the information or data flow thrcugh the system completely suing data flow diagrams, activity diagrams, state transition diagrams etc,. If there is any interruption in the data flow then there is some flaw with the design.

Q.4 Describe code review process.

Ans.:
- The reviewing of the code files, database schemas, classes, object definitions, methods, procedures etc., are defined as code review process.
- It is applied to ensure that the design has been correctly implemented by the developers in the code.
- It must have the following characteristics :
  - **Clarity :** The developer must follow correct coding standards, guidelines, formats as mentioned by the organisation. Code commenting with date, name, revision number etc. must be done by the developer.
  - **Complete :** The code must be complete with respect to class, procedure method, proper indentation etc. It must be readable and compilable.
  - **Traceable :** The code must be traceable with design components. If the code files have no traceability with design it can be considered redundant code.
  - **Maintainable :** The code must be maintainable from future point of view by the coder.

Q.5 Describe unit testing process.

Ans.:
- Unit Testing is a level of software testing where individual units/ components of a software are tested.
- The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software.
- It usually has one or a few inputs and usually a single output.
- In procedural programming, a unit may be an individual program, function, procedure, etc.

- In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/ child class. (Some treat a module of an application as a unit. This is to be discouraged as there will probably be many individual units within that module.)
- Unit testing frameworks, drivers, stubs, and mock/ fake objects are used to assist in unit testing.

Q.6 Differentiate between debugging and unit testing.

Ans.:

	Debugging	Unit testing
1.	It is done to locate causes of the defect.	It checks the defect and not the causes of defect.
2.	The code may be updated during debugging.	It does not involve any correction of code.
3.	Test cases are not defined in debugging.	Test cases are defined in unit testing.
4.	It generally covers positive cases.	It covers both positive and negative cases.

Q.7 What is module testing?

Ans.:

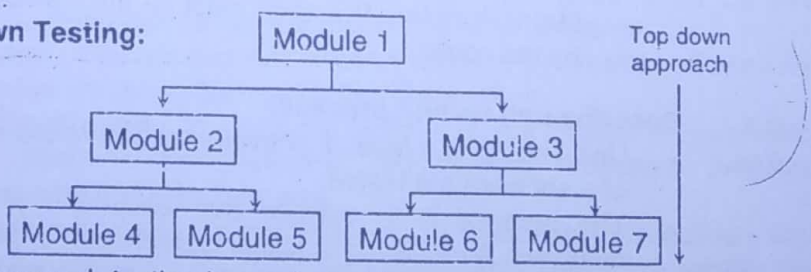
- A module is formed by collection of small units.
- The module may need stubs and drivers to work or may work on its own.
- Module testing focuses on the structure of the system.
- It is done on related unit –tested components to check whether they work together as a module or not.
- The test cases of module testing must be traceable to the requirements or design.

Q.8 Explain integration testing and various approaches of integration testing.

Ans.:

- Integration Testing is defined as a type of testing where software modules are integrated logically and tested as a group.
- A typical software project consists of multiple software modules, coded by different programmers. Integration Testing focuses on checking data communication amongst these modules.
- It is sub divided into Top Down Approach, Bottom Up Approach and Modified top down approach. This process is carried out by using dummy programs called Stubs and Drivers.

Top down Testing:

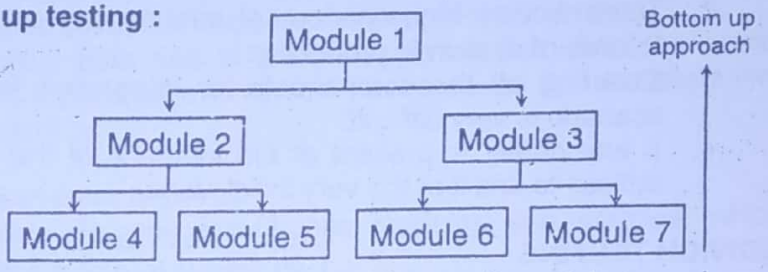


In this approach testing is conducted from main module to sub module. if the sub module is not developed a temporary program called STUB is used for simulate the submodule.

Advantages :
- Advantageous if major flaws occur toward the top of the program.
- Once the I/O functions are added, representation of test cases is easier.
- Early skeletal Program allows demonstrations and boosts morale.

Disadvantages :
- Stub modules must be produced.
- Stub Modules are often more complicated than they first appear to be.
- Before the I/O functions are added, representation of test cases in stubs can be difficult.
- Test conditions may be impossible, or very difficult, to create.
- Observation of test output is more difficult.
- Allows one to think that design and testing can be overlapped.
- Induces one to defer completion of the testing of certain modules.

Bottom up testing :



In this approach testing is conducted from sub module to main module, if the main module is not developed a temporary program called DRIVERS is used to simulate the main module.

Advantages :
- Advantageous if major flaws occur toward the bottom of the program.
- Test conditions are easier to create.
- Observation of test results is easier.

Disadvantages :
- Driver Modules must be produced.
- The program as an entity does not exist until the last module is added.

Q.9 Difference between stubs and drivers.

Ans.:

	Stubs	Drivers
1.	Stubs used in Top Down Integration Testing.	Drivers used in Bottom Up Integration Testing.
2.	Stubs are used when sub programs are under development.	Drivers are used when main programs are under development.
3.	Top most module is tested first.	Lowest module is tested first.
4.	It can simulate the behaviour of lower level modules that are not integrated.	It can simulate the behaviour of upper level modules that are not integrated.
5.	Stubs are called programs.	Drivers are the calling programs.

Q.10 Explain the process of big-bang testing with advantages and disadvantages.

Ans.: In the Big Bang Testing, all the modules or components are integrated before and after this everything is tested as a whole. In this type of testing all units or components are linked at once which result into a complete system. It gives us a complete system after integration which on which we can start testing.

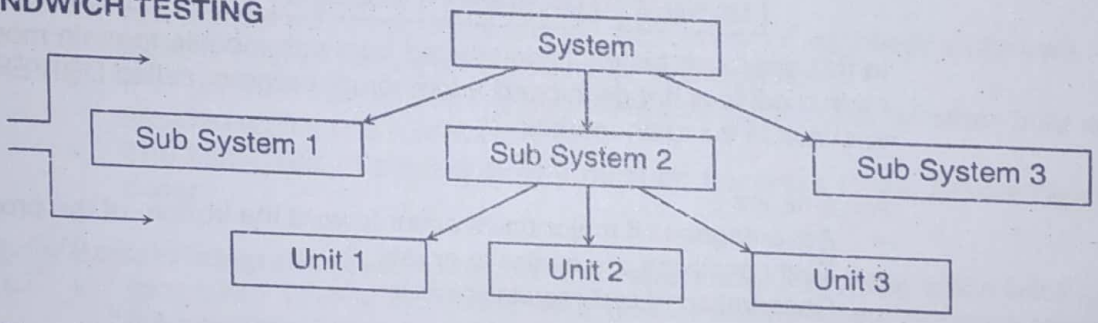
Advantage :

1. Big-Bang Testing is very fast and cheap.
2. No need of help of any middle components like stubs, driver on which testing is dependent.
3. In Big Bang Testing everything is finished before integration testing starts.
4. No need of intermediate builds and efforts is required for the system.

Disadvantage :

1. In Big Bang Testing, it is difficult to trace the cause of failures.
2. There is very high chances of missing any critical difficult which result in failures of system in production.
3. Covering all the components for integration testing without missing any scenario is very difficult.
4. If any defect is present at the interface of the component, then it is very difficult to find it at the very initial stage.

SANDWICH TESTING



Q.11 Explain the process of sandwich testing with advantages and disadvantages.

Ans.:

- Sandwich testing is a type of testing that consist of two parts, they are Top-down approach and Bottom-up approach.
- It combines the advantages of both Bottom-up testing and Top-down testing at a time. Bottom-up testing starts from middle layer and goes upward to the top layer whereas Top-down testing starts from middle layer and goes downward.
- Big-bang approach is followed for the middle layer. From this layer bottom-up approach goes upwards and top-down approach goes downwards.

Advantages :

- Sandwich approach is useful for very large projects having several sub projects. When development follows a spiral model and the module itself is as large as a system, then one can use sandwich testing.
- Both Top-down and Bottom-up approach starts at a time as per development schedule. Units are tested and brought together to make a system. Integration is done downwards.

- It needs more resources and big teams perform both bottom-up and top-down methods of testing at a time or one after the other.

Disadvantages :

- It require very high cost for testing because one part has Top-down approach while another part has bottom-up approach.
- It cannot be used for smaller system with huge interdependence between different modules. It makes sense when the individual subsystem is as good as complete system.

Q.12 What is critical path first concept?

Ans.:

- The critical path indicates the main function of a system which is represented by the P1 requirements (highest priority requirements).
- Testing defines the critical path of the system which must be covered first.
- The development team focuses on the execution of the critical path first.
- This is also called as 'skeleton development and testing'.
- This critical path first is generally performed when the complete system testing is impossible and systems are so large that complete testing may not be economical.

Q.13 What is subsystem testing?

Ans.:

- Subsystem are collection of units, modules, sub modules which have been integrated to form subsystems.
- The testing of such subsystem is called subsystem testing.
- It is also termed as 'integration or interface testing' when individual modules are as good as independent systems.
- It focuses on finding integration or interface errors where one subsystem is supposed to communicate with another subsystem.

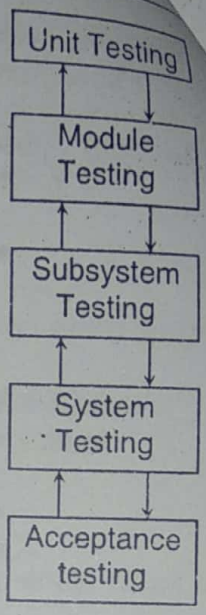
Q.14 Explain stages involved in system testing.

Ans.:

- The final testing done on the system before it is delivered to the customer is called as system testing.
- It is done on the integrated subsystems that make up the entire system or the final system. Following are the stages system testing goes through.
- Functional testing :
  - The function testing is performed to check whether all the functions are working as per requirements or not.
  - Once the functionalities are tested, the defects related to functions can be also fixes to make the system work properly.
- User Interface Testing :
  - The next phase involves setting the user interfaces correct as per requirements.
  - This may involve colours, navigations, spellings and fonts. There are few systems where there is no or minimal user interface, in this case user interface testing may not be applicable.
  - Depending upon the scope of the testing and after functionalities and user interfaces are tested the system can be tested for load, security and performance.

Q.15 Explain the stages of testing with a suitable diagram.

Ans.:



- The initial or first phase of testing is Unit testing, where inputs from unit testing are used to identify and eliminate defects at unit level.
- After Unit testing, once the units are ready for integration, then integration or module testing is done. It can be done by either development team or test team depending upon the requirements of stubs and drivers.
- Once module testing is done, subsystem testing phase starts where the subsystem interaction is tested with other subsystem.
- Once the system completes all levels of integration, it goes for system testing, where complete system is tested.
- Finally system is taken for acceptance testing where it is tested to see whether it fulfills the acceptance criteria of users or not.

Q.16 Explain advantages and disadvantages of GUI / UI Testing.

Ans.:

Advantages :

- Good GUI improves look and feel of the application. It helps in psychological acceptance of the application by the user.
- GUI represents a presentation layer of an application.
- It helps an application due to better experience of the users.
- Availability of 'help' and usefulness of 'help' routines can be ensured so that user can access 'help' in case of problems.
- Consistency of screen layouts and designs improves usability of an application. Tab sequence provides a logical way of doing the sequence.

Disadvantages :

- When the number of pages is large and number of controls in a single page is huge, it creates problem in testing. 'Looking but not seeing' phenomenon may be witnessed in GUI testing, where spelling mistakes may go unnoticed due to fatigue on part of the tester.
- Special applications testing like those made for blind people or kids below age of five may need special training for testers, as they have to behave like target users.
- GUI in terms of images shown on the screen like medical and diagnostic application may be very difficult to test, if testers do not have sufficient domain expertise.
- Testers may give more importance to functionalities than GUI testing. Defects in GUI are considered as cosmetic defects and neglected many times.
- Other GUI testing is considered as low level of testing and given to junior testers.

Q.17 Explain Compatibility Testing.

Ans.: Compatibility Testing refers to testing the software on multiple configurations to check the behaviors of different system components and their combinations.

The variables can be,
- Operating systems
- Databases
- Browsers
- Languages

The hardware can be,
- Machines and servers
- Routers
- Printers

Integration with other communication systems can be,
- Mailing softwares
- Messaging software

Different languages used across the globe can be,
- Chinese, Korean or Japanese
- Hindi, Urdu or Hebrew
- English, German or French

It is performed to ensure that the application functions properly on multiple system configurations which are possible at user end. More compatibility helps in increasing the market size and number of customers for product organisations. Matching more and more criteria can be a challenging task as designs and implementations becomes very complex.

Q.18 What is security testing?

Ans.: Security testing is a special type of testing intended to check the level of security and protection offered by an application to the users against unfortunate incidences like loss of privacy, loss of data, etc.

Some definitions associated with security are given below :
- **Vulnerability :** The weaker parts of the system represent the vulnerabilities in the systems. There is no system in existence which does not have any vulnerability. One must take precaution not to expose these weak points to outsiders.
- **Threats :** Threat represents the possible attacks on the system from outsiders with malicious intentions. Threat is defined as an exploitation of vulnerabilities of the system.
- **Perpetrators :** Perpetrators are the entities who are unwelcome guests in the system. They can create a problem in a system by doing something undesirable like loss of data and making changes in system. Perpetrators can be people, other systems, viruses, etc.
- **Points of Penetration :** The points where the system can be penetrated or where the system is least guarded represented the point of penetration. These points represent the vulnerabilities in the system.

Q.19 Explain the process of security testing.

Ans.: The process of security testing is given as follows :

- Make a list of all possible perpetrators of the system. This may include the people using the system, internet cloud if any, and possible attacks like hacking, viruses, etc.
- Make a list of all penetration points for the system where the attack is likely to happen. Attack can be on different layers such as presentation layer, logic layer and database layer.
- Make a penetration matrix with one dimension as perpetrators, and another dimension as a point of penetration. Each quadrant will give the possible failure point of a system.
- Define the probability of security breakage, and impact of such breakage for each failure point represented by each quadrant. Each quadrant has probability and impact associated with it. Product of probability and impact represents the risk associated for the application. Higher score or greater RPN / RIN represents more problematic situation. Such cases must be taken for protection first during development. They also indicate the probable areas where testing must be concentrated.
- Execute tests on the basis of high–risk prioritisation which is a product of probability and impact or PRN / RIN.

Table : Security matrix / Penetration matrix

	Perpetrators			
Points of Penetration	P × I	P × I	P × I	P × I
	P × I	P × I	P × I	P × I
	P × I	P × I	P × I	P × I
	P × I	P × I	P × I	P × I

Q.20 Explain performance, load and stress testing.

Ans. : Performance Testing :

- Performance testing is done to find whether the system meets its performance requirements under normal load or normal level of activities.
- Normal load must be defined by requirement statement.
- Performance criteria must be expressed in numerical terms. Design verification can help in determining whether required measures have been taken to meet performance requirements or not.
- Performance criteria must be measurable in quantitative terms such as time in 'milliseconds'. Some examples of performance testing can be as given below :
  - Adding a new record in database must take maximum five milliseconds. It means that when the record is added in database, it may take time lesser or equal to five milliseconds.
  - Searching of a record in database containing one million records must not take more than one second. One must add one million records in the system, and then use the search criteria to test this.
  - Sending information of say one MB size across the system with a network of 512 KBPS must not take more than one minute. User may have to try various combinations in this testing.

Volume (Load) Testing :

- Volume testing is about the maximum volume or load a system can accept before it collapses due to load or volume. It can be described in terms of concurrent users, number of connections, maximum size of an attachment to be transferred over a network, etc.
- Volume testing needs some kind of automation when the requirements are very stringent. One may have to increment the load by smallest possible factor and conduct many iterations till one finds a point where system actually collapses.

Example : Simple transactions are created and the number of users is incremented till the system fails. Transaction selected must represent very high probability of happening.

Stress Testing :

- Stress Testing is about the system resources required by the transactions that have been planned to be undertaken. If the system has limited resources available, the response of the system may deteriorate due to non–availability or loading of resources.
- Stress testing is used to define the resources level required by the system for its efficient and optimum performance.

Example : Simple transactions are created, and the system resources such as processor size, RAM size, and bandwidth are reduced to find the minimum level of resources where system is barely living.

Q.21 Explain Recovery Testing.

Ans.: Recovery testing is done to find how the system as a whole, or an individual machine/server as a component of entire system recovers from a disaster.

System Recovery :

There are three famous ways of disaster recovery of a system.

(a) **System Returns to the point of Integrity after meeting Disaster :** When system recovers from the disaster, it comes to the point of integrity last known to the system and user may have to start from that point onwards. The definition of point of integrity differs from system to system.

(b) **Storing Data in Temporary Location :** When the system meets with a disaster, the transactions made till that point are stored temporarily by the system. As soon as system returns to its original state, user is shown the last transactions and asked to confirm whether transactions must be committed or not. If user accepts the transaction, it may be recorded in the system and user can progress from that point onwards. If user rejects it, transaction is lost permanently and user may continue to use the system.

(c) **Completing the transaction :** When the system needs with a disaster or stops working, the transactions upto that point are automatically committed. The system will try to complete the committed transaction till the point upto which it is possible to proceed and stops at that point.

Machine Recovery :

- Machine such as application server and database server may contain very vital information and data. It is very important that data should not be lost when the system meets with disaster.
- Machine recovery is of four types, as given below :
  - (a) **Cold Recovery :** Cold recovery talks about backing up of a data at a defined frequency such as once in a week. The data is backed up on an external device like tape/CD. Backup media may be kept at a location as defined by backup plan.
  - (b) **Warm Recovery :** Warm recovery happens when a backup is taken from one machine to another machine directly. Frequency of backup may be more frequent and automatic backup is possible as it is machine-to-machine backup.
  - (c) **Hot Recovery :** Hot recovery represents a scenario where two machines, original as well as backup machine, are present in the system. One machine is a primary machine while the second machine is treated as a backup machine or standby machine. Both machines are of same or similar configurations and capabilities. Backup frequency is defined as per backup plan. When a problem occurs with one machine, controls are shifted to second machine containing backup data so that it can be used.
  - (d) **Mirroring :** Mirroring also involves two machines with same configuration and backup frequency is mirroring data, i.e., every millisecond data is copied from one machine to another. The data is backed up oniine. If something happens to primary machine, then backup machine takes over the primary machine immediately without any manual intervention.

Q.22 Describe installation, uninstallation and upgradation test.

Ans.: Installation testing is to find how the application can be installed by using the installation guide or documentation given for installation along with installation media like CD.

Installation Process :

- Installation may be done through devices like CD, pen drive and floppies etc.
- Installation may have some prerequisites which are essential for installation or working of software. During installation, it must identify the presence or absence of these prerequisites, and if they are not available. It must inform the user about it.
- Installation process can be fully automatic or it may need user actions as the case may be.

Uninstailation Testing :

- Un-installation must clean all the components and files installed during installation. It must not leave any thread that the installation was done on that system or machine.
- Un-installation Testing can be a requirement of product where the product removed from the system must clean the system completely. The process of un-installation testing may be as follows :

  One may have to take an image of a hard disk before installing software. This can be used to note all the files existing at the time of installation. Then,

the tester must install the application and again capture the image of hard disk. One must compare two images to find if any pre-existing file has been overwritten during installation. Then, one may uninstall the application and take an image of hard disk again. This image must be compared with the first image before installation of software. These two images must match exactly if clean un–installation is proved.

Upgradation Testing :

- Upgradation may be done by using patches released by the product manufacturer from time to time. It may be done using CD, floppies or any other media used for upgradation. Process of upgradation may be as follows:

  During upgradation, the installer must be able to identify that there is an older version of same application available on the disk. No upgradation is possible when there is no application existing on the disk. Also, if existing application is more updated than the upgradation available, no upgradation is possible.

Q.23 Describe Requirement Testing process.

Ans.:

- A requirement statement is needed for testing requirements.
- The tester is expected to write the business case using requirement statement.
- He/she must identify the actors in the business case, and transactions done by different actors during execution of these use cases.
- At any place when decisions are involved, all possible outcomes of the decisions must be covered in requirements. Any outcome not covered gives incompleteness to requirements.
- Test cases are written by referring to the requirement statement. These test cases are executed in system testing. If any defect is found, it is noted and fixed during defect fixing phases.

Q.24 Explain Regression Testing and list development methodologies where it is very important.

Ans.:

- Regression testing determines whether the changed components have introduced any error in unchanged components of the system.
- The maintenance projects may consider it as special testing as regression testing of entire application may be very costly.
- It can be done at,
  - Unit level to identify that changes in the units have not affected its intended purpose, and other parts of the unit are working properly even after the changes are made in some parts.
  - Module level to identify that the module behaves in a correct way after the individual units are changed.
  - System level to identify that the system is performing all the correct actions that it was doing previously as well as actions intended by requirements after change is made in some parts of the system.
- Regression testing is performed where there is high risk that changes in one part of software may affect unchanged components or system adversely. It is done by rerunning previously conducted successful tests to ensure that unchanged components function correctly after the change.

- Following are the development methodologies where regression testing is important :
  - Any maintenance activity conducted in a system.
  - Iterative development methodology.
  - Agile development.

Q.25 Explain Error handling testing.

Ans.:
- If anything unexpected happens with system, application is exists to help user by showing error messages.
- Error handling has a direct relationship with usability of an application, and a distant relationship with security of system.
- System may give various error messages when something wrong is being tried by the user, or system does something wrong while processing data entered by the user.
- It is done to determine the,
  - Ability of system to properly process erroneous transactions and protect the users from making any mistake during data entry.
  - All expected errors by application system are recognised and the appropriate error messages are given to the users when an error happens.
  - Procedures must provide that high–probability errors will be detected and corrected properly before system processes such data.

Q.26 Explain types of error messages.

Ans.: There may be several types of error messages such as the following :
- **Preventive Messages :** When user tries to enter some wrong data, system identifies a wrong entry and prevents such entry in system.
- **Auto–Corrective Message :** It is mainly used to prevent the user from reentering the transaction, where there is a single way of entering the transaction and second time, it may not be accessible to user for correction.
- **Suggestive Message :** The system tells the user about what is wrong and provides suggestions about correcting it.
- **Detective Message :** Detective messaging happens when system tells the user about what is wrong but there is no guidance available about what can be the correct transaction.

Q.27 Explain Manual Support Testing.

Ans.: Manual Support Testing is intended to test the interfaces between users of an application and application system.

Manual testing is performed to determine whether,
- Manual support procedures are sufficiently documented, complete and available to user.
- Manual support people are adequately trained to handle various conditions of working including entering data, processing, taking outputs as well as handling errors. People must be able to work with system independently.
- Manual support and automated segments are properly interfaced within the application. Help available must provide guidance to common user when some problem is encountered by them.

- User must be able to work with system without assistance of system personnel.
- Provide inputs to support group, and enable manual support group to enter into the system at proper time or when users need their help.

Q.28 Explain intersystem testing.

Ans.:

1. Testing of interfaces between two or more systems is called as intersystem testing.
2. It is done to make sure that they work correctly and information is transferred between different systems. System testing helps to determine following :
   (i) Parameters and data are correctly passed between application and other systems to avoid any communication failure.
   (ii) Documentation for the involved system must be accurate, complete and matching expected inputs and outputs. It must define parameter passing and bridge of communication between various systems.
   (iii) System testing must be conducted whenever there is a change in the parameters between applications communicating with each other.
   (iv) Representative set of test transactions is prepared in one system and passed on another system for processing and the results are verified for correctness.
   (v) Manual verification of documentation is done to understand the relationship between different systems.

Q.29 Explain control testing.

Ans.: Control Testing is conducted to verify data validity, file integrity, audit trail, backup and recovery and documentation for the system under development.

Control system is done to determine the following :

- Data processed is accurate, complete and can be used by the normal users. Users must complete all mandatory fields before saving the record.
- Transactions entered in the system are authorised by identifying the user permissions. Unauthorized person may not be able to access the records, or may not be able to save it or modify it or delete it.
- Ensure integrity of processing of transactions and data from entry till exit. Data must not be lost, modified or added during the transaction processing.
- Identify risks associated with different users using the system, and their ability to interact with the system. Access right definitions must be followed.
- Create risk conditions in the test laboratory to access the level of control mechanism. Subject system to these conditions for testing to validate the control provisions in system designs.
- Evaluate effectiveness of controls defined by the requirements. Controls must be effective, efficient and must justify their presence.

Q.30 Difference between Smoke testing and Sanity testing.

Ans.:

	Smoke Testing	Sanity Testing
1.	It is done test basic functionality of software application, to ensure application is living.	It is done to test the major functionality of the application.

2.	The depth of smoke testing is less than sanity testing.	The depth of sanity testing is more than smoke testing.
3.	Steps involved in smoke testing can be installation, navigating through application etc.	It demonstrates connectivity to database, application servers etc. basic GUI Testing.
4.	Smoke testing is not applied for testing the application but to ensure user will be able to work with it.	Sanity Testing is applied for testing the application.

Q.31 What are the advantages and disadvantages of adhoc testing?

Ans.: Adhoc testing is performed without any formal test plan, test scenario, test cases or test data. It is also called 'exploratory testing' or 'monkey testing' or 'random testing'.

Disadvantages :

- This testing may not be reproductive as tester may not remember all the steps done till the point where the defect has been seen. Sometimes, defect is not reproducible only because tester does not remember the steps.
- The scenario testing may not be definable, and may not represent real-life business scenario. Some adhoc scenario with an intention to break the system may not have much value in real life. Sometimes, probability of happening of such events may be next to '0'.
- Adhoc testing needs testers with very good domain expertise and good command over testing process.

Advantages :

- The scenarios tested are adhoc, and there may not be a particular sequence. System may be under stress while executing such scenarios.
- It may try some scenarios which may not be considered at all while writing requirement statement.
- It may need less time as there is no test plan, test scenario and test cases to be written. It is also termed 'playing with an application'.

Q.32 Explain Parallel Testing.

Ans.: Parallel testing is done by comparing the existing system with newly designed system to validate it against existing system.

Parallel testing is done to determine whether,

- The new system performs correctly with reference to existing system that is supposed to be working correct.
- There is consistency / inconsistency between two systems. Consistency may be mainly in terms of user interactions, user capabilities, etc. Consistency with respect to processing of transactions and controls designed for validation is also evaluated.
- It is used extensively in business piloting or beta testing while accepting a new system. This is also called 'comparison testing' where old system behavior is considered as correct.
- New system is used in parallel with the existing system for certain time period, to find the differences between two systems.

Q.33 Explain Execution Testing.

Ans.: It is performed to ensure that system achieves desired level of proficiency in production environment when normal users are using it in normal circumstances. Execution testing involves actual working on the system in production environment to determine whether.

- The system meets its design objectives as defined in requirements and expected by users.
- Execution testing is used to evaluate users experience with new system.
- System must be used at that point of time when results can be used to modify system structure, if required. Alpha and Beta testing anomalies may be used to modify system, if required.
- Execution testing may be conducted by using hardware / software monitors, by simulating the functioning of the system, and by creating programs to evaluate performance of completed system.

Q. 34 Explain Operations Testing.

Ans.: Operations testing is performed to check that operating procedures are correct as documented in user manuals and staff can properly execute the application by using the documentation. It is done to find out the following :

- People must be able to work with the system by referring to documentations of operator, user manual etc.
- User training is complete and user can use the system on the basis of training provided. Effectiveness of training may be evaluated in the testing.
- It must occur prior to placing the application into production environment.
- Actual users must be capable of working with system and system must be able to work with normal users.
- It must be conducted without any assistance provided to operators, as if it was part of normal computer operations.

Q.35 Explain Compliance Testing.

Ans.: The compliance testing is done to check whether system is developed in accordance with the prescribed standards, procedures and guidelines applicable to them as per domain, technology, customer etc. It helps to determine whether,

- Development and maintenance methodologies are followed correctly or not.
- Domain related protocols are followed correctly or not.
- Completeness of system documentation with respect to applicable standards is ensured. Documentation must be clear and complete and must be complying with standards applicable.
- Compliance depends upon managements desire to have standards enforced as well as geographical and political environment where application will be working. Customer requirements must include definition of regulatory and statutory requirements.
- Compliance testing is done using the following :
  - Checklist prepared for evaluation or assessment of product
  - Peer reviews to verify that the standards are met
  - SQA reviews by quality professionals
  - Internal audits

Q.36 Explain Usability Testing.

Ans.: Usability testing involves using user guides and help manuals (including online help) available with application by normal user to find its usefulness. It is applied to determine whether,

- It is simple to understand application usage through look, feel and support available like online help.
- It is easy to execute an application process from user interface provided. If training is required for using the application, it must be provided to users who will be using it.

Usability Testing is done by,

- Direct observation of people using the system, noting their interactions with system and case with which these users can work with the system under testing.
- Conducting usability surveys by checking the deployment or implementation of system in production environment.
- Beta testing or business pilot of application is user environment.

Q.37 Explain Decision Table Testing.

Ans.:

- The strength of decision table testing is that it creates combinations of conditions.
- It may be applied to all situations when the actions of the software depending on several logical decisions are occurring at same time.

Table : Purchased volume and discount

No. of Books	1–50	51–500	501–5000	5001 and more
Discount offered	0%	2%	3%	5%

- The above table indicates single dimension of variable. In practical situations, there can be multiple dimensions possible.
- Axiom testing is a part of decision table where system works on some relationships between variables. When somebody is testing a combination of two variables X an Y which are related to each other with a relationship expressed as Y = f (X), it may be termed 'axiom testing'.

Q.38 What is Documentation Testing.

Ans.:

- Documentation testing involves review of all the documentation accompanying sources /executable to the customer so that system can be maintained in future. All the artifacts must be in sync with each other and must represent a system which is being delivered.
- The purpose of documentation testing is to thoroughly go through all written material that will be presented to the user as a part of implementation. The testing needs to be done in a few different ways.
  - Ask users to follow all documented procedures. These should be written to provide step–by–step guidance on how to accomplish a given task. If the users cannot successfully complete the procedures, the documentation needs to be improved.
  - Have people with strong language skills to review all the documentation for professionalism and readability.
  - Try out all alternative ways that are documented to accomplish a task. In many cases, the primary way works, but the alternatives do not work as expected.

- If one is describing policies or standards, then make sure that the appropriate authorities in the organization review and approve them. It would be disastrous to misquote or misapply an important company policy.
- Evaluate any manual forms, checklists and templates to ensure that they are accurate and the appropriate information is being collected.

Q.39 What is Training Testing?

Ans.:
- The vendor is expected to give training to users who will be using the system in future.
- A better approach is to test training as a part of system testing. This implies that the training must be ready at this point in the life cycle and not created at the very end of the project.
- In the case of distance learning, one must make sure that the technology used is correct for the purpose. One must make sure that there are adequate equipments for imparting such training.
- Training material may be a deliverable of the project. It needs to be tested for accuracy and defects, before it is rolled out for the first time.

Q.40 What is Rapid Testing?

Ans.:
- Rapid Testing is used when there is too little time available to obtain full test coverage using conventional methodologies.
- Rapid testing finds the biggest bugs in the shortest time, and provides the highest value for money.
- In most development projects, there are a number of critical times when it is necessary to make an instantaneous assessment of the products quality at the particular moment.

Some areas where rapid testing is applied extensively are given below :
- 'Proof of concept' test early in the development cycle.
- Prior to, or following migration from development to the production environment.
- Sign–off of development milestones to trigger funding or investment.
- Prior to public release or delivery to the customer.

Working :

Rapid testing is based on exploratory testing techniques, which means that the tester has a general test plan in mind but is not constrained by it. The plan can be adapted on–the–fly in response to the results obtained for previous tests. The drawback is that it is not possible to guarantee total test coverage, but the benefit is that a skilled tester can quickly find faults that would have eluded a scripted test.

Rapid testing extends the exploratory concept by making judgement about what faults to report and the level of details to be recorded. Once a fault has been identified, the time taken to investigate and document it reduces the time available to find other faults, so the tester may fail to find the serious faults if they spend too much time reporting less important issues, we refer to this as the 'quality threshold' and it is fundamental to the effectiveness of rapid testing.

Q.41 What is a Control Flow Graph?

Ans.: A flow of a control when a program is getting executed is called Control Flow Graph. Whenever there is any decision to be taken by a program, there is a

possibility of different flow graphs possible. Each decision induces multiple paths in a program while it is getting executed.

- Dominators : There exists a set of code which will always be executed when a path is selected, then it is termed 'dominator'.
- Post–Dominator : From any point in an application if we go to the end of the application, then if we have to go through particular set of code, it is defined as 'post–dominator'.

Q.42 Explain combinatorial Test Design Process.

Ans.: The combinatorial Test Design Process involves the following steps :

- **Modeling the Input Space and Test Environment :** The model consists of set of factors and corresponding levels. Factors are decided on the basis of interaction between application and environment.
- **Generate Combinatorial Object :** It is an array of factors and levels selected for testing. Such an array will have each row covering atleast one test configuration from the list.
- **Generate Tests and Test Configurations :** The test cases and test configurations are generated from combinatorial object.

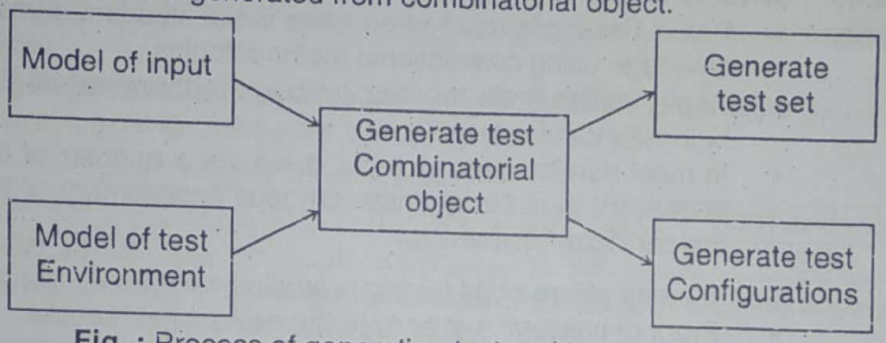


Fig. : Process of generating test and test configurations using combinatorial designs

Q.43 Define State Graph.

Ans.:

- A state graph is useful models for describing software behavior under various input conditions.
- State testing approach is based upon the finite state machine model for the structures and specifications of an application under testing.
- In a state graph, states are represented by nodes.
- Whenever an input is provided, the system changes its state. This is also called 'state transition'.
- When there are multiple state graphs, it becomes difficult to draw them.
- To remove this difficulty, state tables are used. State tables are defined as follows :
  - Each row indicates a state of an application.
  - Each column indicates the input going to an application.
  - Intersection of rows and columns indicate the next state of application.

❑❑❑❑❑